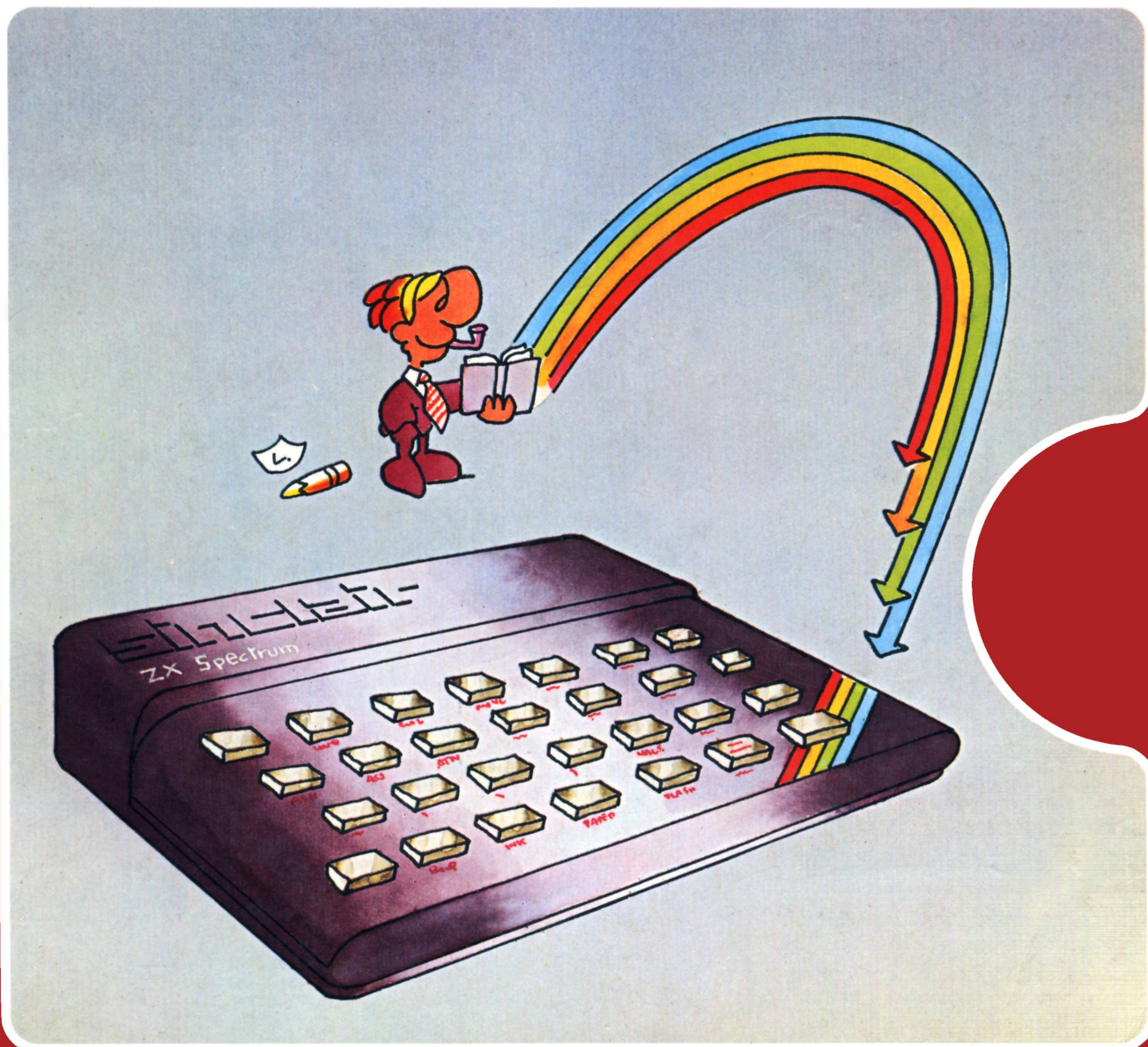
IAN STEWART ROBIN JONES

IL PIACERE DI PROGRAMMARE CON LO ZX SPECTRUM

dal BASIC alla grafica e alla musica
con uno dei più popolari home computer

Edizioni Elettroniche Serie LIBRI

I libri delle Edizioni Elettroniche VIFI-Mondadori si rivolgono al pubblico degli utenti, attuali o potenziali, del personal computer e, in genere, a quanti sono interessati agli sviluppi, alle idee e alle tecnologie della cultura informatica e della società dell'informatica.

Volumi pubblicati:

Giulio Banfi

FACILE COME IL BASIC
guida al linguaggio di programmazione più diffuso sui personal computer
(5ª edizione)

R. Bradbeer P. De Bono P. Laurie

CAPIRE IL COMPUTER
come sono fatti e che cosa possono fare gli elaboratori

Aldo Cavalcoli

SCEGLIERE IL PERSONAL COMPUTER
(3ª edizione)

Alberto Cazziol Roberto Galimberti
Marco Maiocchi Roberto Polillo

UN PROGRAMMA CHIAMATO DOSSIER®
applicazioni di dossier elettronici per la gestione del lavoro d'ufficio

Vincenzo De Rosso

COME SI PROGRAMMANO I COMPUTER
(4ª edizione)

Richard E. Pattis

PROGRAMMARE CON KAREL IL ROBOT
l'informatica con carta e matita

Horacio C. Reggini

LOGO: ALI PER LA MENTE
il linguaggio di programmazione ideato per l'educazione e il gioco creativo

Fausto Servello

CHE COS'È LA TELEMATICA
le nuove tecnologie della società dell'informazione
(2ª edizione)

Mario Trovato

IL CALCOLATORE TASCABILE NELLA SCUOLA
un'introduzione all'informatica
(2ª edizione)

Fuori collana:

Luca Novelli

IL MIO PRIMO LIBRO SUI COMPUTER
(2ª edizione)

Luca Novelli

IL MIO PRIMO LIBRO DI BASIC

IAN STEWART ROBIN JONES

IL PIACERE DI PROGRAMMARE CON LO ZX SPECTRUM

dal BASIC alla grafica e alla musica con uno dei più popolari home computer

ARNOLDO MONDADORI EDITORE

Indice

Prefazione

Clive Sinclair ha di nuovo fatto centro!

Non pago di aver venduto oltre mezzo milione di microcomputer ZX81, ha lanciato sul mercato lo ZX Spectrum. Non solo è ora possibile avere un calcolatore domestico a un prezzo contenuto: potete anche disporre di suono, colore e grafica ad alta risoluzione (e di una tastiera di buona qualità!).

Lo scopo di questo libro è semplice: descrivere in termini chiari e comprensibili quelle caratteristiche dello ZX Spectrum che un utente principiante dovrebbe conoscere. Inoltre abbiamo incluso circa 30 listati di programmi "preconfezionati" da copiare e far "girare" a vostra discrezione; più qualcosa come altri 30 programmi nel corso del testo. Il nostro vantaggio sul *Manuale* è che possiamo essere selettivi, concentrarci sugli aspetti più importanti tralasciando dettagli che potrebbero far perdere il senso generale del discorso.

Così cosa si ottiene? Si ottiene una facile guida alla programmazione in BASIC, allo stile di programmazione, al colore, al suono, alla grafica animata, alla (fantastica) grafica ad alta risoluzione, alla correzione dei programmi (*debugging*), all'elaborazione numerica, alle operazioni sulle stringhe. Dopo aver letto questo libro, il *Manuale* sarà un bocconcino. Ogni argomento è trattato in sezioni autonome, in modo da permettervi di volta in volta di passare un paio d'ore con lo Spectrum ottenendo dei risultati concreti. In più troverete una varietà di programmi ricreativi, che fanno buon uso della notevole gamma di possibilità dello Spectrum, per divertirvi quando l'attenzione inizia ad affievolirsi.

Moltissimi libri sui calcolatori danno l'impressione di cercare di dimostrare quanto intelligenti siano i loro autori. Non è il nostro caso: qui tutto è diretto ed essenziale.

Fino a questo punto abbiamo fatto riferimento a noi stessi come "noi", al plurale, ma questo non succederà in seguito. Le esperienze personali riferite con un "noi" assumono un aspetto curioso. Dal momento che ogni singolo capitolo è stato scritto da un solo autore, abbiamo votato per la parola "io" per descriverci entrambi. Se ciò vi reca disturbo, considerate quanto spesso un singolo autore alluda a se stesso come "noi"!

1. Le basi del BASIC

ROM, RAM, e REM... il gergo dei computer

Non c'è niente di fondamentalmente misterioso riguardo ai computer. Sono solamente macchine che portano a termine serie di istruzioni. Naturalmente il *modo* in cui effettivamente le eseguono può essere veramente misterioso, a meno che non siate laureati in fisica dello stato solido. Tuttavia, proprio come potete guidare un'auto senza saperla costruire, potete programmare un calcolatore senza essere in grado di progettarne uno. Una certa conoscenza del reale funzionamento delle cose — ciò che gli addetti ai lavori chiamano l'*hardware* (lett. "ferraglia") — è comunque utile, proprio come è d'aiuto sapere come funziona una frizione quando state imparando a cambiare marcia, se non addirittura indagare perché sia proprio necessario cambiare marcia. Ma i principi fondamentali della programmazione non dipendono molto dall'hardware, per quanto concerne un utente comune.

Macchine che compiono istruzioni sono tra noi da lungo tempo. Blaise Pascal, il matematico e filosofo francese, costruì una macchina per calcolo nel 1642.

L' inglese Charles Babbage progettò un ambizioso "motore analitico" nel 1835, e il governo britannico finanziò la sua costruzione. Come altri progetti governativi, si rivelò non realizzabile con la tecnologia del tempo; ma le idee che ne erano alla base erano originali e valide. Il francese Joseph-Marie Jacquard, all'inizio del diciannovesimo secolo, sviluppò un sistema che utilizzava dei fori punzonati su schede per controllare l'orditura prodotta da un telaio. Gli organetti a vapore delle fiere usano un principio simile. In realtà si può pensare a un'orchestra ben affiatata come a una macchina per la produzione di musica, con la partitura musicale come "programma".

Non è opportuno andare da un telaio Jacquard, dire "sette metri di tweed a scacchi blu e verdi, prego!" e aspettarsi che succeda qualcosa di molto utile. Non parla quella lingua. Tutto quello capisce è una disposizione di fori punzonati; e c' è ogni sorta di restrizioni su dimensioni, numero e ordinamento di quei fori. Il nastro perforato che suona una canzone sull'organo non funzionerebbe su un telaio — e anche se funzionasse, produrrebbe un risultato molto poco so-

migliante al motivo originale. È necessario sapere *quale* disposizione di fori sia necessaria per produrre il disegno desiderato sul tessuto. Così il disegno dei fori è una versione *codificata* del disegno dell'ordito, con il meccanismo del telaio che agisce da tramite.

In modo simile, la musica scritta, utilizzata da un'orchestra, è una versione codificata dei suoni desiderati, con l'orchestra stessa come intermediario (certo le orchestre non sono precise come una macchina, e il direttore ha qualche libertà d'interpretazione).

Con i computer è lo stesso: con una data macchina bisogna parlare in una lingua che questa comprenda.

Ai primordi della computazione automatica, questo significava che avreste dovuto scrivere lunghe liste di questo aspetto: 010001110100110101111111100010110001010011... che è piuttosto sgradevole alla vista. Tuttora i calcolatori "in fondo" *pensano* in questo modo (con gli "0" che significano "niente corrente elettrica" e gli "1" che vogliono dire "presenza di corrente elettrica"): questo linguaggio piuttosto rudimentale è il *codice macchina* per un determinato tipo di computer. Miracolosamente, vi permette di fare qualsiasi cosa vogliate; e generalmente ha il vantaggio di essere *rapido* nell'esecuzione.. ma non è semplice scrivere in codice macchina.

Fortunatamente al giorno d'oggi non è necessario.

Questo perché si può *insegnare* a un calcolatore ad accettare comandi in una lingua diversa dalla sua "lingua madre". Ciò che succede in realtà è che qualche diligente programmatore stende una sorta di "dizionario" per tradurre la nuova lingua nel codice macchina e lo fornisce al computer (trasmesso da qualche sorgente esterna, oppure insito nell'hardware). Il dizionario è noto come *compilatore* o *interprete* (a seconda del suo funzionamento).

Lo Spectrum ha un interprete incorporato per un linguaggio noto come BASIC, che sta per "Beginners' All-purpose Symbolic Instruction Code" (linguaggio simbolico di uso generale per principianti). Questo linguaggio è stato sviluppato negli USA alla metà degli anni '60 ed è di uso molto vasto. Ha il vantaggio di essere relativamente semplice, quasi un incrocio tra l'inglese comune e l'algebra scolastica. Per quanto riguarda il programmatore principiante, lo Spectrum parla solo BASIC (tuttavia il linguaggio macchina dello Z80 è accessibile tramite il tasto USR ma se non sapete *perché* potreste volerlo usare, *non* servitevene. Non ancora, in ogni caso).

Un programma

Invece di partire con la "grammatica" del BASIC, diamo un'occhiata a un semplice programma BASIC per vedere cosa fa e come lo fa. In questo modo vedrete immediatamente quanto sia facile. Le sottigliezze grammaticali verranno in seguito.

```
10  PRINT "Raddoppio"
20  INPUT x
30  LET y = x + x
40  PRINT x, y
50  STOP
```

Solo guardandolo, probabilmente riuscite a indovinare che cosa faccia. Ma, in primo luogo, ci sono alcune cose che vale la pena di sottolineare sulla struttura del programma.

(a) Consiste di una serie di *linee*.
(b) Ogni linea è un'*istruzione* o *comando* (queste due parole in pratica significano tutte la stessa cosa) BASIC "legale" (cioè, logicamente sensata).
(c) Ogni linea inizia con un numero, noto (senza sorpresa) come *numero di linea* (i calcolatori usano spesso Ø per indicare lo zero, per distinguerlo dalla lettera O).

Devo aggiungere alcuni chiarimenti a queste note piuttosto ovvie. Alcune di queste valgono per *tutti* gli interpreti BASIC, usati su macchine diverse; altre solo per lo Spectrum. Per risparmiare fiato non mi preoccuperò di distinguere le une dalle altre: se passerete a una macchina più elaborata, scoprirete le differenze in un attimo.

Non c'è niente che valga la pena di dire su (a), eccetto il fatto che per lo Spectrum una linea di programma non corrisponde a una linea scritta sullo schermo TV — può essere più lunga: alla macchina non importa.

I dettagli circa (b) dipendono dalla grammatica del BASIC, e ci arriveremo.

La ragione per numerare le linee (c) è che presto, nella vostra carriera di programmazione, vorrete dire alla macchina di obbedire a una specifica linea di un programma. Un' istruzione "GOTO 730" dirà al calcolatore di fare qualsiasi cosa si dica alla linea 730, e il numero serve proprio per poter individuare quella specifica linea. Come vedremo, ci sono dei vantaggi nel *non* conteggiare solo 1, 2, 3... Il numero di linea deve essere un numero intero tra 1 e 9999 compresi.

Quando la macchina fa il suo lavoro lungo un programma, si muove da una data linea alla successiva in ordine numerico (a meno che non le abbiate detto, come parte del programma, di andare altrove). Così ha bisogno di quei numeri di linea, anche quando lo stesso programmatore vorrebbe farne a meno. Lo Spectrum non accetta linee di programma senza numero (sebbene possa invece eseguire gli stessi comandi che compaiono in quella linea, se introdotti in *modo diretto* da tastiera, come se fosse una calcolatrice tascabile).

Non dimenticate i numeri di linea!

Che cosa fa?

Introducete il programma precedente nel vostro Spectrum. Ho intenzione di spiegare tutte le complicazioni della tastiera un poco più avanti: avrete certamente notato che ogni tasto reca una terribile quantità di scritte e, infatti, alcuni tasti possono avere sei effetti diversi a seconda della situazione di "mode" e "shift"! Ma non avrete bisogno di conoscere la dura verità tutta in un colpo.

Ammetterò che abbiate già acceso la macchina, e ottenuto il messaggio "© 1982 Sinclair Research Ltd". (In caso contrario, premete NEW o staccate la spina di alimentazione per un secondo).

Prendiamo la linea 10 come esempio. Premete il tasto 1 (fila in alto, a sinistra) e quindi il tasto 0 (fila in alto, a destra). Vedrete que-

sti numeri apparire sul lato inferiore dello schermo, con una "K" lampeggiante, detta *cursore*. Premete, poi, il tasto P: appare l'intera parola "PRINT". È questa una caratteristica molto intelligente dello Spectrum, e risparmia un sacco di tempo: intere parole BASIC possono essere introdotte usando un solo tasto (avrete notato che anche la parola "PRINT" è scritta sul tasto P).

Ora le virgolette ". Sono anche queste sul tasto P! Ma in *rosso*. Questo significa che per ottenere le virgolette dovete tenere premuto il tasto SYMBOL SHIFT mentre premete il tasto P. Ora rilasciate SYMBOL SHIFT. Per ottenere una R maiuscola, tenete premuto il tasto CAPS SHIFT mentre premete il tasto R. Lasciate CAPS SHIFT e premete, in sequenza, i tasti A, D, D, O, P, P, I, O. Queste lettere verranno stampate in minuscolo: "addoppio". Ora di nuovo le virgolette: SYMBOL SHIFT e P come in precedenza.

La linea 10 è ora sul lato inferiore dello schermo (inoltre il cursore è diventato una "L" lampeggiante). Per inserirla in memoria, premete il tasto marcato ENTER. È salita verso l'alto!

La linea 20 si scrive nello stesso modo: premete 2, 0, I (che reca la scritta "INPUT"), e X; quindi ENTER. Le linee da 30 a 50 sono simili; solo notate che si ottiene il simbolo "=" tenendo premuto SYMBOL SHIFT mentre si preme L ; il segno "+" usando SYMBOL SHIFT e K; la virgola con SYMBOL SHIFT e N; lo STOP è SYMBOL SHIFT più A. Non dimenticate ENTER, una volta che una linea è stata scritta in modo corretto.

Se avete seguito queste istruzioni, avrete l'intero programma "listato" sulla parte superiore dello schermo. E rimarrà lì per sempre, a meno che non facciate qualcosa per evitarlo. Questo perché i calcolatori non solo obbediscono a delle istruzioni, ma sono fondamentalmente ottusi: dovete in ogni caso dire loro quello che volete che facciano anche quando ciò è perfettamente ovvio. Così premete il tasto R (che sta per RUN) e quindi ENTER, in modo da fargli capire che avete finito il comando.

Quello che vedrete sullo schermo sarà qualcosa come RUN [L] con una L lampeggiante. Quest'ultima sparisce quando premete ENTER.

Sullo schermo appare il messaggio "Raddoppio". Questa è la risposta del computer alla linea 10: PRINT "Raddoppio". L'esecuzione è stata piuttosto rapida, e ora il calcolatore sta attendendo che *voi* facciate la *vostra* parte della linea 20: INPUT x (inserire x). Per rammentarvelo, c'è una [L] nell'angolo inferiore sinistro dello schermo. Vuole che gli diciate che cosa è x.

Perciò, scrivete un numero, seguito da ENTER. Provate con

2 (ENTER)

Rapido come un fulmine il calcolatore stampa

2 4

(e, nell'angolo inferiore, un messaggio "9 STOP Statement, 50:1" di cui non mi curerò — significa solo che il compito è stato portato a termine correttamente).

Riprovate. Premete dunque RUN, seguito da ENTER. Quando domanda x, provate qualcosa d'altro: se battete 756.2912 e premete ENTER, dovreste ottenere

```
756.2912        1512.5824
```

Fatelo "girare" ancora qualche volta, con diversi valori di x in ingresso. Provatene qualcuno a caso. Provate 0, 1, 2, 3, 4.

Questo dovrebbe convincervi che, qualsiasi cosa introducete come x, la macchina stampa due numeri: prima x, e quindi il doppio di x.

Sperimentate. Dovreste ottenere qualcosa come

```
 17     34
 21     42
—5    —10
  6     12
```

Come lo fa?

Non intendo addentrarmi in dettagli — questo ci porterebbe all'hardware, al linguaggio macchina, e a simili complicazioni. Ma come voi, programmatori, dovreste considerare la macchina, quando esegue il vostro programma? Che cosa sta succedendo dentro il suo piccolo cervello di silicio? Indulgendo in un poco di antropomorfismo, avviene qualcosa del genere.

10 PRINT "Raddoppio" 20 INPUT x 30 LET y = x + x 40 PRINT x, y 50 STOP	Bene, qui ci sono delle istruzioni che devo ricordare. Meglio farlo. Mi domando se ha già finito.
RUN ENTER	Via. Andiamo. Qual è la prima linea? Prendiamola dalla memoria: 10 PRINT "Raddoppio". Devo stampare qualcosa. Ci sono le virgolette "; allora copio quello che segue...
Raddoppio	finché non trovo delle altre ". Questo mi dice di smettere di stampare. Nient'altro. Avanti alla prossima linea: 20 INPUT x. Sta per dirmi un numero che devo chiamare x. Diamogli un...
[L]	cursore per ricordarglielo. OK, sto aspettando. Incredibilmente lenti, questi umani.

2 ENTER

Ah. x è 2. E ora?
30 LET y = x + x. Ciò significa che devo sommare x a x, cioè, eseguire 2 + 2.
E chiamare il risultato y.
Così y è 2 + 2 = 4.
La prossima istruzione?
40 PRINT x, y.
Devo stampare x e y, che sono...

2 4

La linea successiva? 50 STOP.
Allora è tutto. Termino con un rapporto...

9 STOP Statement, 50:1

Fantasie a parte, vedete come la macchina stia solamente muovendosi lungo la lista dei comandi, e obbedendo loro quando si presentano? Non "sa" nemmeno di che cosa tratti il programma. Ma voi, i programmatori, lo sapete — raddoppia dei numeri.

Il compito del programmatore è ora chiaro. Dato un compito che volete la macchina vi porti a termine, dovete costruire una serie di istruzioni che, una volta eseguite, ottengano questo risultato.
Per questo, vi occorre una conoscenza più dettagliata del BASIC. Il vostro Spectrum potrà fare molte cose, se saprete come parlargli. Naturalmente un *buon* programma dev'essere rapido, efficiente e chiaro, oltre che ottenere il suo scopo. Ma le raffinatezze possono aspettare: la cosa più importante è scrivere programmi che *funzionino*.

Cominciamo!

2. Aritmetica in BASIC

Addizione, sottrazione, moltiplicazione e divisione: le basi della matematica valgono anche per il calcolo automatico

In realtà il titolo di questo capitolo è falso — si tratta di algebra, ma non volevo impressionarvi. Praticamente ogni programma che vi interessi scrivere (anche solo per questioni contabili), comporta calcoli di tipo algebrico, così questi rappresentano un argomento ragionevole per iniziare.

Proprio come la comune algebra, il BASIC utilizza lettere per indicare numeri "generici". In gergo, queste sono dette *variabili* — per meglio dire, variabili *numeriche* — ma in realtà tutto questo significa che si tratta di cose come x, y, z, a, b, c, eccetera, che si possono utilizzare per scrivere espressioni algebriche come x + y — z, che il computer può eseguire se gli dite, tramite il programma, che valori assumono x, y e z (per esempio se x = 14, y = 3, z = 9, allora x + y — z = 14 + 3 — 9 = 8). Con lo Spectrum si possono usare per le variabili simboli più complicati che non singole lettere, ma nomi lunghi sprecano memoria.

Esistono delle piccole, ma importanti differenze tra l'algebra BASIC e quella comune. I segni + (più), — (meno), e / (diviso) sono proprio come i soliti (non si possono però usare i due punti per la divisione). Ma la moltiplicazione viene scritta con un asterisco *, così 5 * 7 significa 5 × 7, che è uguale a 35. La freccia che punta in alto ↑ significa "elevazione a potenza". Per esempio 2 ↑ 3 indica quello che un algebrista scriverebbe come 2^3, cioè 2 alla terza potenza (o 2 *al cubo*), e vale 2 × 2 × 2 = 8.

Si possono combinare numeri, variabili e questi segni aritmetici per produrre espressioni più complesse.

Per esempio, a * x ↑ 2 + b * x + c corrisponde, nella sua formulazione più usuale, ad $ax^2 + bx + c$.

Il Capitolo 3 del libro d'introduzione allo Spectrum spiega queste cose e sottolinea l'importanza dell'ordine in cui la macchina esegue le operazioni. Nell'espressione precedente, supponiamo che la macchina sappia che a = 4, b = 3, c = 7, x = 5. Si potrebbe pensare

che eseguirà la somma come segue:

$a * x = 4 * 5 = 20$

$a * x \uparrow 2 = 20 \uparrow 2 = 400$

$a * x \uparrow 2 + b = 400 + b = 403$

$a * x \uparrow 2 + b * x = 403 * x = 2015$

$a * x \uparrow 2 + b * x + c = 2015 + c = 2022$

È quello che si otterrebbe risolvendo l'espressione da sinistra a destra. Ma questo non corrisponde a

$4 \times 5^2 + 3 \times 5 + 7 = 100 + 15 + 7 = 122$

Che cosa non è andato per il verso giusto?

Il fatto è che la comune algebra è zeppa di regole su quale operazione fare per prima. Il linguaggio BASIC ha regole simili. Infatti esegue tutti gli ↑ *per primi*, poi tutti gli * e /, e alla fine i + e —. In caso di dubbio, esegue tutte queste operazioni nell'ordine da sinistra a destra. Così in realtà esegue un $a * x \uparrow 2 + b * x + c$ nel modo seguente:

Prima ↑	$x \uparrow 2 = 5 \uparrow 2 = 25$
Poi *	$a * x \uparrow 2 = 4 * 25 = 100$ $b * x = 3 * 5 = 15$
Poi +	$a * x \uparrow 2 + b * x = 100 + 15 = 115$ $a * x \uparrow 2 + b * x + c = 115 + c = 115 + 7 = 122$

Si può vedere come questo ricalchi la comune algebra.
Proprio come in algebra, capita di voler eseguire un'espressione che *non* segue queste regole in modo naturale. La soluzione è la stessa: usare delle *parentesi* ( ). Qualsiasi espressione all'interno delle parentesi viene portata a termine per intero per prima. Così

$(a * x) \uparrow 2$

verrebbe eseguita come $(4 * 5) \uparrow 2$, cioè $(20) \uparrow 2$ se si fa per prima la parte tra parentesi, e si ha 400.

Esiste un solo tipo di parentesi (non ci sono le [ ] e { } dell'algebra). Le altre parentesi sono presenti sulla tastiera, ma non si possono usare nelle espressioni. Dovete quindi essere particolarmente attenti quando inserite più livelli di parentesi, scrivendo $(3 + 5 * (a - b)) \uparrow 4$ dove un algebrista scriverebbe $[3 + 5 (a - b)]^4$.

La cosa principale da ricordare è che in realtà tutto funziona come nella solita algebra, con una simbologia un poco meno familiare.

Assegnazione di un valore a una variabile

Vediamo ora come funziona l'istruzione di assegnazione LET (tasto L). Se la usate dove dovrebbe essere usata, lo Spectrum la riconosce automaticamente come LET e non come L (per mezzo di un proce-

dimento noto come "controllo automatico della sintassi", che significa che la macchina tiene d'occhio ciò che ha senso nell'istante in cui lo battete sulla tastiera). Una linea di programma come

```
40 LET x = 5
```

dice al computer che *da questo istante in avanti, e finché non disposto diversamente da qualche altra parte del programma*, la variabile x assume il valore 5. Provate questo programma

```
10  LET a = 4
20  LET b = 3
30  LET c = 7
40  LET x = 5
50  PRINT a * x ↑ 2 + b * x + c
```

Se tutto funziona bene, vedrete apparire in alto sullo schermo la risposta: 122.

Esistono metodi più semplici per ottenere questa particolare risposta: provate

```
10  PRINT 4 * 5 ↑ 2 + 3 * 5 + 7
```

(Potete persino omettere il numero di linea; consultate il Capitolo 3 del libro d'introduzione allo Spectrum. Ma noi volevamo solo vedere come funziona LET).

Il secondo termine di un comando LET può essere un'espressione che il computer sappia già calcolare. Per esempio, potreste modificare la linea 50 per definire una nuova variabile y, che assuma il valore a * x ↑ 2 + b * x + c :

```
50  LET y = a * x ↑ 2 + b * x + c
```

Fate seguire

```
60  PRINT y
```

e potete controllare se funziona in modo corretto.

Esempio: corpi in caduta libera

Secondo Galileo, un corpo che cada partendo da uno stato di quiete sotto l'azione della gravità, coprirà una distanza di (circa) 4.9 t^2 metri in un tempo di t secondi. Per trovare quanto spazio percorrerà in un tempo di 17 secondi, potreste usare il programma

```
10  LET t = 17
20  PRINT 4.9 * t ↑ 2
```

Dovreste ottenere in uscita 1416.1 (metri).

Se volete la risposta per un tempo diverso, potete modificare la prima linea del programma. Provate. Quanto spazio percorrerà il corpo in:

(a) 97 secondi?
(b) 3 secondi?
(c) 24.5779 secondi?
(d) 100001 secondi?

Un miglioramento: l'istruzione INPUT

Probabilmente, nell'esempio precedente, avrete fornito i dati con ripetute modifiche della linea 10. Un programma meno rozzo avrebbe impiegato il comando INPUT, che permette alla macchina di chiedervi quale valore assuma una variabile. Scrivete il programma

```
10 INPUT t
20 PRINT 4.9 * t ↑ 2
```

e fatelo "girare". Appare il cursore [L]: la macchina sta aspettando che le diciate quanto vale t. Inserite 17, premete ENTER; otterrete lo stesso risultato di prima. Ma ora, per avere i risultati da (a) fino a (d), tutto quello che dovete fare è far ripartire il programma (RUN) e introdurre il nuovo valore di t. Provate.

Per maggiori informazioni su INPUT, consultate il Capitolo 5. *Ingresso/Uscita*.

Realizzate dei programmi

Provate ora a scrivere i programmi per i tre problemini che seguono, usando i comandi che abbiamo già visto. Dovrete solo apportare delle piccole modifiche al programma dei corpi in caduta. Se siete nei pasticci, cercate aiuto alla fine di questo capitolo.

(e) Il volume di un cubo di lato x è dato da x^3 (cioè x ↑ 3 in BASIC). Scrivete un programma che vi permetta di attribuire diversi valori a x e che dia come risultato il volume del cubo corrispondente.

(f) Un secchio è appeso all'estremità di una corda, avvolta attorno a un verricello cilindrico di raggio r. Secondo i testi di meccanica, per questo particolare secchio, l'accelerazione diretta verso il basso è data dalla formula

$$a = \frac{32}{1 + 3r^2}$$

Senza preoccuparvi della meccanica, scrivete un programma che vi permetta d'inserire r e che stampi l'accelerazione a.

(g) Allo Spectramarket Zedex, il miele costa 1500 lire il vasetto, i dadi 650 lire il pacchetto e la Supercarne Whirxas 900 lire la scatoletta. Scrivete un programma che stampi la spesa totale, per m vasetti di miele, d pacchetti di dadi e s scatolette di carne, quando inserite i valori di m, d, s. Vi serviranno tre comandi di INPUT.

Un cenno sui cicli

I comandi FOR e NEXT ci permettono di raffinare alcune delle idee precedenti senza lavorare troppo (in molti circoli, l'evitare il lavoro si chiama ''pigrizia'', ma i programmatori hanno un vantaggio sui comuni mortali — possono chiamarlo ''programmazione efficiente'' e puntare a risparmiare tempo e memoria del computer). Ci sono molte cose da dire sui cicli FOR/NEXT, e tra poche pagine vi dedicheremo tutto lo spazio necessario. Per prima cosa però fate girare questo programma, per vedere che tipo di cose si possono fare.

```
10  FOR x = 1 TO 20
20  PRINT x, x * x
30  NEXT x
```

Otterrete una lista di numeri da 1 a 20 e una lista corrispondente dei loro quadrati 1, 4, 9 ... 361, 400. Quello che fanno i comandi FOR/NEXT è spedire la macchina continuamente lungo la serie di comandi che sono tra loro interposti (in questo caso solo la linea 20) ponendo la variabile x successivamente a 1, 2, 3... finché non si raggiunge 20. La linea 10 stabilisce i valori estremi della variabile e fa partire il ciclo; la linea 30 rispedisce la macchina lungo il ciclo.

Modificando la linea 20 in questo programmino, potrete tabulare di tutto. Volete i cubi? Provate

```
20  PRINT x, x ↑ 3
```

Riscrivete la linea 20 nelle diverse varianti che seguono. I piccolissimi cambiamenti, tra l'una e l'altra, faranno sì che la macchina esegua i calcoli algebrici in diverso ordine, con diverso risultato. Che cosa stanno calcolando i vari programmi, nel comune simbolismo algebrico?

```
(h)  20  PRINT x, 1/x + 1
(i)  20  PRINT x, 1/(x + 1)
(j)  20  PRINT x, 1/1 + x
(k)  20  PRINT x, 1/(1 + x)
(l)  20  PRINT x, 1 + 1/x
(m)  20  PRINT x, (1 + 1)/x
```

Infine notiamo che in questi programmi i valori della variabile non sono assegnati dal comando LET. La macchina li ottiene invece dal valore attuale della variabile corrente.

Risposte

(a) 46104.1

(b) 44.1

(c) 2959.9585

(d) 49000980000 (espresso in notazione esponenziale). Non lasciatevi ingannare dall'apparente precisione. Il calcolatore non dà un'accuratezza di 11 cifre significative. Ha arrotondato il vero risultato, che è 49000980004.9.

(e)
```
10 INPUT x
20 PRINT x ↑ 3
```

(f)
```
10 INPUT r
20 PRINT 32/(1 + 3 * r ↑ 2)
```

(g)
```
10 INPUT m
20 INPUT d
30 INPUT s
40 PRINT 1500 * m + 650 * d + 900 * s
```

Non è necessario che il vostro programma sia esattamente come questi, per essere ''corretto''. In particolare, la scelta dei simboli per le variabili dipende interamente da voi. Potete anche realizzare la parte algebrica in modi differenti: per esempio, alla linea 20 di (e) potreste scrivere PRINT x * x * x.

(i) $\frac{1}{x+1}$

(h) $\frac{1}{x} + 1$

(j) 1 + x [esegue prima 1/1, uguale a 1, poi somma x]

(k) come (i)

(l) come (h)

(m) $\frac{2}{x}$

3. La tastiera

Una rapida guida per insegnarvi a distinguere i diversi significati che ogni tasto può avere

Come ho detto in precedenza, la tastiera dello Spectrum è piuttosto sofisticata, e il risultato dell'azionamento di ogni singolo tasto dipende dal contesto in un modo abbastanza complicato. Vi ci abituerete presto; ma vi servirà un po' di tempo per imparare a padroneggiare compiutamente la tastiera.

Le due cose che influenzano il significato di un tasto sono lo *shift* utilizzato e il *modo*, cioè lo stato in cui si trova il computer.

Tasti SHIFT

Ci sono due tasti marcati CAPS SHIFT e SYMBOL SHIFT. Se uno di questi viene tenuto premuto mentre si aziona un altro tasto, l'effetto di quest'ultimo si modifica. *Come* esattamente, dipende dallo stato della macchina, come stiamo per vedere.

I modi

Il modo è lo "stato interno" del calcolatore, e viene segnalato dal cursore. Esistono cinque modi, corrispondenti ai seguenti cursori (lampeggianti):

K modo parole-chiave (*keyword*)
L modo lettere (*letters*)
C modo maiuscole (*capitals*)
E modo esteso (*extended*)
G modo grafico (*graphics*)

Ecco come arrivarci:

1. La macchina si trova normalmente nel modo "L".
2. Dopo un numero di linea, o all'inizio di un comando impartito direttamente, va automaticamente nel modo "K".
3. Per arrivare nel modo "C" dal modo "L", premete CAPS SHIFT e tenetelo premuto. Per tornare nel modo "L" rilasciatelo (vedrete solo la "C" se usate CAPS LOCK — blocco maiuscole — per il momento non preoccupatevene).
4. Per ottenere il modo "E" dal modo "L", premete *sia* CAPS

SHIFT *sia* SYMBOL SHIFT. Per tornare nel modo ''L'', ripetete l'operazione.

5. Per arrivare nel modo ''G'' dal modo ''L'', premete CAPS SHIFT e il tasto 9 (GRAPHICS) insieme. Per tornare nel modo ''L'', ripetete l'operazione.

Come stampare quello che volete

Esaminiamo un tipico tasto in una delle tre file inferiori, diciamo il tasto ''R''. Ha questo aspetto:

INT è in verde; < e VERIFY in rosso.

1. Per scrivere ''r'', premete il tasto quando siete nel modo ''L''.
2. Per scrivere ''R'', premete CAPS SHIFT e andate nel modo ''C''; tenetelo premuto e azionate il tasto R.
3. Per scrivere ''RUN'' premete il tasto quando vi trovate nel modo ''K''.
4. Per scrivere ''<'' andate nel modo ''L''; tenete premuto SYMBOL SHIFT e azionate il tasto R.
5. Per scrivere ''INT'' andate nel modo esteso, ''E'', premendo contemporaneamente entrambi gli SHIFT. *Controllate il cursore ''E''*: se tenete premuti i tasti troppo a lungo può tornare a ''L''. Rilasciate gli SHIFT e premete R.
6. Per scrivere ''VERIFY'' andate nel modo ''E'' come in (5) e *tenete premuto SYMBOL SHIFT* mentre premete R.
7. Per ottenere il carattere grafico d'utente che, una volta definito, corrisponde a ''R'' (leggete in proposito il Capitolo 13) andate nel modo ''G'' e premete il tasto R. (Se non è ancora stata operata la definizione del carattere d'utente, verrà stampata una ''R'').

La fila di tasti superiore

Questi tasti presentano alcune differenze. Non recano parole-chiave (modo ''K''): c'è invece un carattere grafico a forma di quadratino. Superiormente non ci sono scritte in verde, ma dei comandi in bianco. Il simbolo principale del tasto e i due simboli rossi funzionano proprio come nelle altre file. Le parti rimanenti si comportano come segue:

1. Per ottenere il simbolo bianco superiore, tenete premuto CAPS SHIFT. Questi sono caratteri di controllo, e non vengono effettivamente *stampati*.
2. Per avere il simbolo grafico, andate nel modo ''G'' e premete il tasto.
3. Per avere il simbolo grafico in forma negativa (cioè per scambiare tra loro bianco e nero o, più generalmente, INK — inchiostro — e PAPER — carta —) andate nel modo ''G'', tenete premuto CAPS SHIFT, e premete il tasto.

4. I colori sono solo mnemonici: servono a ricordare che, per esempio, 4 è il codice di VERDE.
5. In realtà si può fare qualcosa di più: consultate il *Manuale*. E non ho approfondito le funzioni di CAPS LOCK, TRUE VIDEO e INVERSE VIDEO. Dal momento che in realtà non avrete *necessità* di questi tasti, invece di aumentare la confusione, vi lascerò sperimentare per conto vostro.

Programmi preconfezionati

Nell'ultima sezione di questo libro troverete numerosi listati di programmi. *Dovreste sentirvi in ogni momento liberi di copiarne uno e farlo eseguire*, sia che comprendiate o meno i comandi contenuti nel programma. Tutto ciò che dovete fare è battere il listato sulla tastiera, controllare attentamente per assicurarvi che non ci siano errori, e premere RUN, seguito da ENTER. Noi speriamo che alla fine del libro siate in grado di arguire come questi programmi svolgano il loro compito; ma, per prendere rapidamente confidenza, è una buona pratica provare a eseguire programmi scritti da altri. Ciò vi dà anche un'idea di che cosa si possa far fare allo Spectrum.

I programmi sono generalmente accompagnati da Note di Programma che spiegano diverse particolarità, e occasionalmente suggeriscono modifiche o progetti da tentare in settori correlati. Ma i programmi funzioneranno, salvo errori di battitura, indipendentemente dal fatto che queste note abbiano per voi senso o meno. La maggior parte dei programmi illustrano tecniche particolari spiegate nel corso di questo libro, ma alcune fanno uso di idee che non ho modo di esporre debitamente. In ogni caso presumo che stiate leggendo il *Manuale* Sinclair, oltre al presente testo.

4. Aiiuuutoooo!

Non va bene come speravate? Eccovi qualche consiglio

Cosa dovete fare, se c'è qualcosa che non va proprio bene?

Nel caso peggiore, staccate l'alimentazione per un secondo. Questo frantuma il programma, spazza via tutto ciò che avete scritto — ma elimina i disturbi, qualsiasi sia la loro origine (tranne che sia un problema di hardware, caso di sfortuna nera). Ma solitamente ci sono dei metodi migliori.

(a) Se scrivete qualcosa di sbagliato in una linea di programma: usate i tasti 5 e 8, contrassegnati da una freccia (notare che serve CAPS SHIFT), per portare il cursore [L] subito dopo il carattere (o la parola-chiave) indesiderato, cancellatelo usando DELETE (tasto 0 con CAPS SHIFT).

(b) Volete modificare una linea che è già inserita nel programma: per cancellarla del tutto, scrivete il suo numero di linea e premete ENTER. Altrimenti muovete il cursore > su o giù lungo le varie linee usando 6 e 7: poi premete EDIT per spostare la linea in basso, dove si può modificare. Procedete come in (a). *Trucchi vari*: è terrorizzante muovere il cursore dalla linea 10 alla linea 530, una linea per volta. Trovate un numero di linea che non avete usato, giusto prima di 530, ad esempio 529. Scrivete 529 ENTER; poi premete EDIT. Vedrete scendere la linea 530. (Perché?).

(c) Se il programma si blocca e sembra che non accada più niente: è spesso questo il caso che si verifica durante le operazioni di messa a punto del programma; l'ultima cosa che desiderate fare è cancellare e reinserire tutto e, anche così facendo, non è improbabile che si ripresenti esattamente lo stesso problema.

(c1) Se il programma è ancora in esecuzione, ma volete fermarlo: premete CAPS SHIFT e BREAK.

(c2) Se è bloccato su un'istruzione di INPUT numerico: BREAK non funzionerà; ma STOP sì. Se c'è qualche altro pasticcio continuerà a darvi un cursore [?] come per segnalare un errore di sintassi, cosa estremamente frustrante; spazzate via i "rifiuti" con DELETE, poi usate STOP.

(c3) Se è bloccato su un INPUT di caratteri — cursore [L] : premete DELETE e *poi* STOP.

(d) Se il programma procede, ma non fa quello che dovrebbe: leggete i capitoli sul *Debugging*.

5. Ingresso/Uscita

INPUT/OUTPUT: ci sono una quantità di metodi utili per inserire ed estrarre informazioni dal calcolatore

Ingresso

Nel comando INPUT c'è molto più di quanto ho detto finora. Per cominciare, potete inserire più numeri in una sola volta. Un modo più elegante per risolvere l'esercizio (g) del Capitolo 2 è:

```
10  INPUT m, d, s
20  PRINT 1500 * m + 650 * d + 900 * s
```

Quando eseguirete questo programma, scoprirete che dovete premere ENTER dopo *ognuno* dei tre inserimenti. I numeri vengono temporaneamente scritti sulla (o sulle) righe inferiori, finché non sono stati inseriti tutti.

Le virgole che li separano controllano anche il luogo dove i numeri vengono stampati: vanno alternativamente nelle colonne 0 e 16. Se trasformate le virgole in punti e virgola ";", troverete che gli ingressi vengono stampati l'uno dietro l'altro, senza spazi. Per avere gli spazi, dovete definirli nel comando INPUT:

```
10  INPUT m; "□"; d; "□"; s
```

dove il quadratino indica uno spazio.

In ogni INPUT potete inserire dei messaggi che rammentino che cosa si richiede. Per questo dovete inserire il messaggio tra virgolette, come parte della frase di INPUT. Per esempio, modificate la precedente linea 10 in:

```
10  INPUT "miele", m
12  INPUT "dadi", a
14  INPUT "scatolette", s
```

Potete costruire combinazioni più complesse di comandi simili, facenti parte di una sola istruzione INPUT, ma quanto detto è sufficiente per comprenderne le caratteristiche peculiari.

Uscita

Finora, quando abbiamo stampato con una PRINT dei dati in uscita (come dei numeri) abbiamo lasciato alla macchina il compito di decidere *dove* stampare: non sempre però questa soluzione porta ad ottenere dei risultati facilmente leggibili e gradevoli. Per modificare la posizione di stampa, si usa PRINT AT.

Per quanto riguarda una PRINT, lo schermo TV si considera suddiviso in 22 *linee* numerate 0-21 dall'alto verso il basso; e 32 *colonne* numerate 0-31 da sinistra a destra. C'è uno schema, nel Capitolo 12, che illustra la situazione; tuttavia, per il momento, ecco un programma che vi permetterà di fare qualche prova.

```
10  INPUT "linea", l
20  INPUT "colonna", c
30  PRINT AT l, c; "£"
40  GOTO 10
```

Letteralmente stampa moneta! Provate diversi valori per i numeri di linea e di colonna e guardate dove vengono stampate le lire.

Naturalmente, per scegliere una determinata posizione di stampa, specificate i valori l e c nel comando. Per avere un messaggio nella linea più in basso, a partire da cinque spazi dall'inizio (cioè sulla colonna 4, dato che i numeri partono da 0!), scrivete:

```
10  PRINT AT 21, 4; "Messaggio"
```

Per averlo circa a metà schermo:

```
10  PRINT AT 10, 12; "Messaggio"
```

E così via.

Se non utilizzate l'istruzione AT, automaticamente il computer si muoverà solo fino alla posizione "successiva". Dove in realtà, dipende da quello che è stato appena stampato. Se l'ultimo comando PRINT non finisce con un punto e virgola o una virgola, allora la macchina si sposta alla linea seguente. Un ";" la fa andare allo spazio successivo nella *stessa* linea. Una "," la porta alla colonna 0, o alla colonna 16, scegliendo delle due posizioni la prima libera che incontra. Il punto e virgola e la virgola hanno un effetto analogo anche nei comandi INPUT.

Abbinato a queste caratteristiche automatiche, il comando TAB è molto utile se state cercando di stampare dati organizzati per colonne. Per esempio, questo programma vi permette di costruire una rubrica telefonica personalizzata.

```
10  INPUT "Nome"; n$
20  INPUT "Località"; l$
30  INPUT "Numero"; t
40  PRINT TAB 1; n$; TAB 15; l$; TAB 25; t
```

(TAB è il tasto P in modo esteso — E —. Il segno del dollaro indica le *stringhe*, che verranno spiegate in un successivo capitolo loro dedicato). Lanciate l'esecuzione del programma e date degli INPUT come

Carlo	Timbuktu	420396
Bettino	Roma	12345
Polizia	Milano	113

a quanto vi viene richiesto. Vedete come i dati vengano disposti in tre colonne (per fermare il programma, premere STOP durante un input numerico, oppure DELETE e poi STOP durante un input di una stringa).

Progetto

Scrivete un programma per inserire 22 numeri, e stamparli diagonalmente dall'angolo alto a sinistra verso destra in basso (usando qualcosa come PRINT AT i, i; n). Pensate ora a come stamparli da destra in alto verso sinistra in basso (PRINT AT i, 21—i; n).

6. Iterazioni

Erano in 10 nel letto e il più piccolo disse
"Rotolatevi, rotolatevi!"
Si rivoltarono e uno cadde.
Erano in 9 nel letto e...

Le istruzioni FOR/NEXT

Usando le istruzioni FOR e NEXT, potete far sì che la macchina esegua ripetutamente un'istruzione, per un numero definito di volte. Ciò potrebbe non apparire molto interessante, ma in realtà è una delle armi più utili nell'arsenale del programmatore, dato che si possono utilizzare delle variabili per modificare l'azione di ogni singolo passaggio nel ciclo.

Abbiamo già visto come cicli FOR/NEXT ci permettano di stampare tabelle di valori di una funzione come x↑3. Ecco un uso un poco meno semplice.

Come tutti gli studenti sanno, la funzione fattoriale n! è definita come:

$$n! = n \times (n - 1) \times (n - 2) \times (n - 3) \ldots \times 3 \times 2 \times 1$$

Fornisce il numero dei possibili ordinamenti di n oggetti. Per esempio, $3! = 3 \times 2 \times 1 = 6$, e infatti le tre lettere a, b, c si possono ordinare in 6 modi diversi: abc, bac, bca, acb, cab, cba.

Per calcolare n! possiamo utilizzare dei cicli. L'idea è quella di calcolarlo per gradi:

$$1; 1 \times 2; 1 \times 2 \times 3; 1 \times 2 \times 3 \times 4; \ldots$$

continuando finché il numero più grande è uguale a n. In ogni caso prendete il risultato del ciclo *precedente* e lo moltiplicate per il numero intero immediatamente superiore. Questa è proprio la struttura che vi serve per un ciclo. Infatti la sequenza d'istruzioni

```
10  LET i = 1
20  FOR k = 2 TO n
30  LET i = i * k
40  NEXT k
```

ha proprio l'effetto desiderato.

Che cosa fa? E come? La linea 10 predispone un valore iniziale per la variabile i (cioè *inizializza* la variabile). La linea 20 dice alla macchina di ripetere più e più volte le linee successive, facendo sì che k assuma progressivamente i valori 2, 3, 4, 5... n. La linea 40 segnala quando è stato raggiunto il termine di queste istruzioni e impone di ritornare all'inizio del ciclo. Così, al primo attraversamento del ciclo, si prende k = 2 e la linea 30 elabora

$i = i * k = 1 \times 2$

Durante il successivo attraversamento, k è diventato 3 e i 1×2, così si calcola $i * k$, che è ora $1 \times 2 \times 3$. La volta successiva k è 4 e si esegue $1 \times 2 \times 3 \times 4$. E così via, fino all'ultimo stadio, quando k è n e viene calcolato $i = 1 \times 2 \times 3 \times 4 \times \ldots \times n$. Allora, grazie al limite imposto nella linea 20, la macchina sa che il ciclo è finito.

Questo non calcolerà per voi n!, perché il programma non include istruzioni su quanto valga n, o per stampare la risposta. Dovete così inserire il ciclo in un programma più ampio:

```
 5  INPUT n
10  LET i = 1
20  FOR k = 2 TO n
30  LET i = i * k
40  NEXT k
50  PRINT "Il fattoriale di □"; n; "□ vale □"; i
```

(La linea 50 è solo una stampa ricercata: si hanno uscite del tipo

Il fattoriale di 6 è 720

Notate l'uso degli spazi, indicati con i quadratini).

Coniglicoltura

Intorno al 1220, il matematico pisano Leonardo Fibonacci avanzò un interessante problema sui conigli.

Se una coppia fertile di conigli produce una progenie di una nuova coppia una volta al mese, ed è necessario un altro mese perché la nuova coppia diventi produttiva, quale sarà la crescita della popolazione a partire da una sola coppia? (Per semplicità assumeremo una produzione esattamente regolare ogni mese, e che ogni coppia consista di un maschio e di una femmina).

Una tabella può essere d'aiuto.

Al mese 0 abbiamo una coppia fertile, e 0 nuove coppie, ottenendo come dati in ingresso:

Mese	Coppie fertili	Nuove coppie
0	1	0

Al mese 1 abbiamo una sola nuova coppia:

Mese	Coppie fertili	Nuove coppie
1	1	1

Al mese 2 abbiamo una nuova coppia, e quella precedente diviene produttiva:

Mese	Coppie fertili	Nuove coppie
2	2	1

Al mese 3 tutte queste sono fertili e sono nate due nuove coppie:

Mese	Coppie fertili	Nuove coppie
3	3	2

E così via:

Mese	Coppie fertili	Nuove coppie
4	5	3
5	8	5
6	13	8
7	21	13

Stiamo cominciando ad avere una notevole quantità di conigli, cosa non proprio sorprendente.

Se assumiamo che il numero totale di coppie di conigli, al mese m, sia f (m), abbiamo che f (0) = 1, f (1) = 2, f (2) = 3, f (3) = 5, f (4) = 8, e così via.

Ragioniamo ora in termini più generali. Supponiamo che al mese m ci siano c coppie fertili e n nuove coppie. Il mese successivo, tutte queste sono divenute produttive, dando c + n coppie *fertili*; a loro volta, le c coppie fertili, ci daranno c *nuove* coppie. Due linee consecutive della tabella prenderanno la forma seguente:

Mese	Coppie fertili	Nuove coppie
m	c	n
m + 1	c + n	c

Questa tabella suggerisce come si possa calcolare f (m) usando dei cicli. L'idea è che, per andare dal mese m al mese m + 1, dobbiamo trasformare il valore c precedente in c + n, e l'n precedente in c.

Il programma che segue svolgerà questo compito.

```
10  LET c = 1
20  LET n = 0
30  INPUT m
40  FOR t = 1 TO m
50  LET b = c
60  LET c = c + n
70  LET n = b
80  NEXT t
90  PRINT "f ("; m;") □ vale □"; c + n
```

Studiandolo troverete che ricalca fedelmente i passi usati per scrivere la tabella. Le linee 10 e 20 stabiliscono i valori iniziali (*inizializzano*) n e c. La linea 30 richiede quale valore m desideriamo per f (m). Le linee da 40 a 80 eseguono un ciclo che costruisce le successive linee della tabella. Notate la linea 50 che ricorda il *vecchio* valore di c, chiamandolo b, per riutilizzarlo nella linea 70. (Non facendo così, la linea 60 modificherebbe c troppo presto, e la linea 70 produrrebbe un valore n errato).

I numeri f (m) sono detti *numeri di Fibonacci*. Se avete notato che le colonne n e c della tabella contengono gli stessi numeri, ma spostati di una linea (perché?), potete osservare che f (m + 2) = f (m + 1) + f (m); cioè ogni numero di Fibonacci è la somma dei due precedenti.

Quanto vale f (14)? f (77)?

Cicli multipli

Meglio ancora — potete inserire cicli all'interno di altri cicli, persino cicli dentro altri cicli dentro altri cicli... (finché ne può contenere la memoria). Vi sorprenderà quanto spesso ciò sia necessario.

Per esempio, supponiamo che vogliate stampare una tabella della funzione fattoriale. Si può usare un ciclo come abbiamo già visto nel Capitolo 2; ma in questo caso vi serve *un secondo ciclo* per calcolare n!. Si ottiene dunque qualcosa del genere:

```
 5  FOR n = 1 TO 20
10  LET i = 1
20  FOR k = 2 TO n
30  LET i = i * k
40  NEXT k
50  PRINT n, i
60  NEXT n
```


Notate che il ciclo FOR/NEXT relativo a n è completamente esterno a quello di k. Bisogna fare così, perché la cosa abbia senso — proprio come con le parentesi. L'espressione [a + (b — 2c)] ha senso, ma non [a + (b — 2c]). Provate a scambiare tra loro le linee 40 e

60, per vedere che cosa succede. Non molto utile vero? Il problema è che lo Spectrum accetterà ed eseguirà dei programmi con cicli inseriti male; naturalmente i risultati non saranno quelli che desideravate. *Non* stampa un messaggio d'errore: fate attenzione!

Step

Scrivendo solo FOR i = 1 TO 20, la macchina *presuppone* che voi desideriate che la variabile i assuma i valori 1, 2, 3, 4, 5..., 19, 20: in altri termini stabilisce che vi muoviate in *passi* (*steps*) di dimensione 1.

Tuttavia non è necessario che sia così. Potete scegliere dei passi di dimensioni diverse tramite il tasto STEP. Per esempio l'istruzione

```
10   FOR j = —3 TO 3 STEP 0.5
```

farà sì che j assuma i valori —3, —2.5, —2, —1.5, —1, —0.5, 0, 0.5, 1, 1.5, 2, 2.5, 3. In modo del tutto simile

```
10   FOR k = 3 TO 4 STEP 0.01
```

passa attraverso i valori 3, 3.01, 3.02, 3.03... salendo di un centesimo per volta, fino a terminare con 3.98, 3.99, 4.

Potete inoltre fare passi all'indietro. Per esempio

```
10   FOR a = 10 TO 0 STEP —1
```

attribuisce ad a la serie di valori 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0. Vi ricorda qualcosa? Lanciate l'esecuzione di questo programma.

```
10   FOR j = 10 TO 0 STEP —1
20   PRINT j
30   NEXT j
40   PRINT "VIA!"
```

Non molto sofisticato, ma...

7. Debugging I

Si dice che la colorita espressione "togliere i bachi" sia nata ai primordi dei calcolatori, quando gli insetti erano soliti passeggiare dentro la macchina e provocare corti circuiti. Di solito, ai nostri giorni, se il computer non va, è colpa del programmatore. Tuttavia per sistemare qualche problema, serve ancora saper qualcosa di... disinfestazione

È una delle dure realtà della vita, che tutti i programmatori imparano molto rapidamente, il fatto che difficilmente i programmi funzionano — almeno la prima volta che vengono mandati in esecuzione. Il lavoro di eliminazione degli errori da un programma è detto *debugging*, ossia disinfestazione, lotta ai "bachi" (*bug* letteralmente significa "piccolo insetto"). È importante adottare un approccio sistematico a questo processo, perché, persino in programmi piuttosto piccoli, la fonte degli errori non è necessariamente facile da trovare: ci sono poche cose più frustranti di un programma contenente un baco inafferrabile.

Per prima cosa diamo uno sguardo ai vari tipi di errori che possono avere luogo. A grandi linee si dividono in due gruppi: *errori di sintassi* ed *errori in esecuzione (runtime)*.

Errori di sintassi

Sono, questi, errori che la macchina può identificare non appena li avete inseriti. Per esempio, supponiamo che io voglia scrivere

```
10  FOR p = 1 — 7
```

credendo, a torto, che il simbolo "—" si possa usare come sinonimo di "TO". (Questo, se volete, è un errore nell'uso della grammatica del linguaggio, da cui il termine "errore sintattico").

Lo Spectrum in questo caso è di particolare aiuto al principiante: permetterà di stampare il simbolo "—", ma si accorgerà che non esiste alcun caso in cui una linea di programma completa possa apparire come 50 FOR p = 1 — 7, e mostrerà un simbolo di avvertimento [?] (per dirvi che c'è un errore di sintassi), e si rifiuterà di accettare la linea finché non venga sostituito il "—" trasgressore. (Molti dei microcomputer più comuni saranno perfettamente felici di permettervi di cospargere un programma di comandi inesatti come questo, e faranno obiezioni solo quando tenterete di farlo eseguire. Tutto questo non importa a un programmatore esperto, ma il principiante tende a commettere numerosi errori del genere: esser-

ne informati immediatamente fa risparmiare un sacco di tempo).

A questo punto alcuni lettori potrebbero essere tormentati da una domanda che potremmo formulare come segue: "Se lo Spectrum sa che la sola cosa che può comparire dopo 1 nel comando 50 è la parola-chiave TO, perché non la inserisce di sua iniziativa?".

La risposta è che non ne sa abbastanza per poterlo fare. Per esempio, ci potrebbe essere un altro numero di seguito, come in

```
50  FOR p = 12 TO 40
```

(Esistono altre possibilità più complesse. Il comando

```
50  FOR p = 1 — 7 TO 5
```

è valido, dato che la macchina calcola automaticamente 1 — 7 ottenendo come risultato —6).

L'errore può essere stampato dal calcolatore prima della fine della linea. Per esempio, scrivete

```
20  LET e * s = q
```

La macchina si aspetta di trovare un "=" (o un'altra lettera o numero) dopo "e", perciò può emettere il segnale [?] davanti a "*"; non prima però che abbiate premuto ENTER.

Per gli errori di sintassi, quindi, la regola è: *cercate il segnale* [?]. Quando questo appare, la ragione sarà solitamente abbastanza ovvia. Se così non fosse controllate la frase che state scrivendo sulla sezione del *Manuale* che vi si riferisce (un tipo di errore che occasionalmente genera perplessità si ha quando accidentalmente si scrive per intero una parola-chiave, invece di utilizzare il relativo tasto. Per esempio potreste scrivere le due lettere "T" e "O" invece di premere il tasto-chiave "TO": la macchina non lo accetterà, benché sullo schermo non ci sia alcuna differenza).

Errori in esecuzione

Quando un programma viene eseguito, vi possono essere numerosissimi tipi di errori differenti. Una volta incontrai una routine di calcolatore che, a causa di un errore piuttosto misterioso, voleva funzionare solo con programmi con un numero pari di caratteri; prima che fosse stabilito qual'era l'errore, si poteva aggirarlo aggiungendo, in qualche punto non pericoloso, un carattere extra a quel programma che si rifiutasse di funzionare!

È più utile fornire esempi specifici di possibili errori, piuttosto che definirli in termini generali. Per cominciare, diamo uno sguardo al programmino che segue:

```
10  FOR p = —5 TO 5
20  LET a = 10/p
30  PRINT a
40  NEXT p
```

Ogni linea è BASIC perfettamente valido, così non saranno segnalati errori di sintassi. Infatti, dopo aver introdotto RUN, il programma inizierà l'esecuzione perfettamente e produrrà le uscite:

—2	[cioè 10/(—5)]
—2.5	[cioè 10/(—4)]
—3.3333333	[cioè 10/(—3)]
—5	[cioè 10/(—2)]
—10	[cioè 10/(—1)]

Ma poi si fermerà con il messaggio d'errore

6 Number too big, 20: 1

Qui il problema è abbastanza chiaro — anche senza consultare il codice di errore sul *Manuale*. Per prima cosa il messaggio indica che la macchina ha obiezioni sulla linea 20. Secondo, questa istruzione è già stata eseguita più volte senza problemi; perciò la questione deve riguardare una delle quantità che *varia*, cioè il valore di a o p. Il valore di p che è appena stato trattato con successo è —1, così il valore corrente di p è 0. In altri termini, la macchina sta cercando di eseguire 10/0, che è infinito (o non definito, secondo i gusti) e quindi non si può maneggiare, per quanto grande sia lo spazio che la macchina riserva alla risposta.

Probabilmente avete raggiunto questa conclusione molto prima di aver portato a termine la (piuttosto laboriosa) analisi di cui sopra; ma sto utilizzando questo semplice esempio per segnalare che *tipo* di indicazioni dovete cercare quando un errore vi ha tratto in inganno.

1. Identificate la linea del problema (il numero dopo la virgola nel messaggio d'errore).
2. Stabilite se questa istruzione sia stata eseguita almeno una volta, prima di produrre il messaggio d'errore. (Se sì, il problema è il particolare valore di una delle variabili all'istante in cui si è avuto l'errore).
3. Usate il messaggio d'errore per avere ulteriori indizi. Nell'esempio questo è "Number too big": "Numero troppo grande". Notate che il messaggio d'errore non dice *esattamente* ciò che è accaduto (cioè non dice "Tentativo di divisione per zero"), così non è sempre sufficiente osservare il messaggio d'errore nella speranza di una spiegazione precisa di quello che non ha funzionato.

Ci sono altre due cose da dire sulla forma dei rapporti d'errore.

Primo: cominciano con un numero o una lettera (in questo caso 6) che è soltanto un identificatore che si riferisce a un articolo nell'Appendice B del *Manuale*. Questo può darvi, o meno, qualche aiuto per decidere che cosa non ha funzionato. In questo caso sul *Manua-*

le si legge "I calcoli hanno dato per risultato un numero maggiore di circa 10^{38}", il che mi sembra non molto diverso dal messaggio "Numero troppo grande"...

Secondo, alla fine del messaggio, dopo i due punti c'è un numero, normalmente 1. Questo si riferisce alle "linee multi-istruzione" dove c'è più di un comando sulla stessa linea. Finora non abbiamo utilizzato questa tecnica: vedi il Capitolo 9, *Decisioni*, per ulteriori dettagli. A grandi linee quello che succede è che potete inserire diverse istruzioni in una singola linea, purché separate da due punti; questo numero finale nel rapporto d'errore vi dice quale di queste sia la colpevole. Così 20: 1 significa "La prima istruzione della linea 20", mentre 20: 3 si riferirebbe alla terza.

Questo sembra (ed è) molto sensato ma, se non siete attenti, si ha possibilità di confusione. Il motivo è che a questo riguardo lo Spectrum considera ogni "interruzione naturale" nella sequenza delle istruzioni (per esempio un THEN) come l'inizio di una nuova istruzione.

Così la linea

```
10  IF p/0 = 2 THEN LET p = p + 1
```

produrrebbe il messaggio

6 Number too big, 10: 1

ma la linea

```
10  IF p = 2 THEN LET p = p/0
```

produrrebbe il messaggio

6 Number too big, 10: 2

perché l'errore è nella seconda parte dell'istruzione.

Per inciso, questo spiega i messaggi in sovrannumero che si hanno quando *niente* va storto. Per esempio scrivete LIST; otterrete il messaggio

0 O.K., 0: 1

che significa

0 Codice di rapporto 0: la macchina ha eseguito quello che le era stato chiesto e non ha incontrato problemi.

O.K. Forma più succinta per quanto sopra.

0 È stata appena eseguita la "linea 0", che è solo un altro modo per dire che il comando non aveva numero di linea.

:1 Era la prima istruzione di quella linea.

Il problema di dividere per zero sorge piuttosto di frequente, e non necessariamente in un modo così ovvio come quello appena descritto. Per esempio

```
30  INPUT p, q, r
40  LET a = (p + q — r * 2)/(5 + (p — r) * (p — r) — 2 * q)
```

provocherà lo stesso problema se si inseriscono i valori 7, 15 e 2 per p, q e r rispettivamente. (Provate!). In generale è buona pratica controllare i divisori per vedere se si annullano, prima di tentare l'operazione. Potremmo riscrivere l'esempio precedente come segue:

```
30  INPUT p, q, r
32  LET d = 5 + (p — r) * (p — r) — 2 * q
34  IF d = 0 THEN PRINT "Non eseguibile": GO TO 30
40  LET a = (p + q — r * 2)/d
```

Il significato dell'istruzione GO TO ("vai a") è, spero, ovvio! La linea 34 è un esempio di linea multi-istruzione.

Problemi

Per terminare, ecco la possibilità di controllare la vostra comprensione di quanto detto finora. Il programma che segue è inteso ad accettare una serie di numeri ed a stampare la loro media.

Se volessimo la media di 2.4, 8.1, 7, e 14 dovremmo inserire:

```
2.4
8.1
7
14
—1
```

Il "—1" finale *non* fa parte dei dati ed è usato solamente per indicare che non ce ne sono altri da inserire. (Tale valore è spesso detto *terminatore* o *tappo* e capirete come funziona osservando la linea 40 del programma che segue).

```
 10  LET s = 0
 20  LET c = 0
 30  INPUT n
 40  IF n < 0 THEN GO TO 100
 50  LET c = c + i
 60  LET z = s + n
 70  GO TO 30
100  PRINT "La media è □"; s/c
```

Nel listato ci sono degli errori tipografici. Vedete se riuscite a correggerli *inserendo il programma così come è,* e provando a modificarlo. Quando avete ottenuto di farlo girare, confrontate la vostra soluzione con la mia leggendo il Capitolo 11.

Buona caccia ai bachi!

8. Numeri casuali

Nello Spectrum c'è anche una certa dose d'imprevedibilità

L'istruzione RND produce un numero ''casuale'' compreso tra 0 e 1, che può valere 0 ma non 1. In realtà non è proprio casuale, ma è un numero *pseudo*-casuale, che si ripete dopo 65537 volte, ma in pratica la differenza non è avvertibile. Dal momento che nessuno sa come sia in realtà un numero ''casuale'', l'uso di ''pseudo'' è a sua volta un po' pseudo.

Potete utilizzare questa funzione in programmi di giochi. Per esempio, per simulare il lancio di un dado: notate che 6 * RND è un numero casuale compreso tra 0 e 6 (6 escluso), così INT (6 * RND) può essere 0, 1, 2, 3, 4 oppure 5, a caso; quindi 1 + INT (6 * RND) può essere 1, 2, 3, 4, 5 o 6 a caso, cioè uno dei numeri sulle facce di un dado. Per estrarre una carta casualmente da un mazzo da 52 potreste fare un gioco simile con 52 * RND, ma vi servirebbe un bel po' di programmazione per trasformare i numeri tra 0 e 51 nei nomi delle carte, come ''fante di fiori''. Si può fare se siete abili.

Potete anche usare i numeri casuali per fare simulazioni statistiche. Ce n'è un bell'esempio nel programma *Dadi del Monopoli*, che troverete nella sezione finale di questo libro.

9. Decisioni

Quando il calcolatore deve fare delle scelte che dipendono da quel che è successo prima

IF/THEN

Questo capitolo tratta istruzioni logiche e *condizionali*. Per esempio, al cinema, se (IF) avete meno di 18 anni, allora (THEN) non vi lasceranno assistere a un film a luce rossa. Si possono imporre condizioni simili per governare il comportamento del computer.

L'istruzione da usare è IF... THEN... — ma è molto importante quello che c'è al posto dei puntini.

Scriviamo un programma che indichi se un qualsiasi numero è pari o dispari.

```
10  INPUT n
20  IF n = 2 * INT (n/2) THEN PRINT ''Pari''
30  IF n <> 2 * INT (n/2) THEN PRINT ''Dispari''
```

Come funziona? I numeri pari sono esattamente divisibili per 2. Così n è pari solo quando n/2 è intero. Ciò significa che prendendone la parte intera si ha ancora lo stesso valore cioè n/2 = INT (n/2). E *questo* è equivalente a n = 2 * INT (n/2). La linea 20 è quindi un modo arzigogolato di dire

```
20  IF n è pari THEN PRINT ''Pari''
```

che ha un senso per *noi*, ma lo Spectrum non sa cosa significhi ''è pari'' finché non glielo si dice nella lingua che capisce.

Solo per puntualizzare questo fatto, esamineremo alcuni casi.

n	22	23	24	25	26
n/2	11	11.5	12	12.5	13
INT (n/2)	11	11	12	12	13
2 * INT (n/2)	22	22	24	24	26

Dovrebbe bastare a convincere il più duro degli scettici.

(È molto utile sapere che, analogamente, k è divisore esatto di n — con n e k numeri interi — se e solo se n = k * INT (n/k)).

Forse l'altra cosa utile da notare è che alla linea 30 il simbolo < > significa "diverso da". I matematici userebbero ≠, ma lo Spectrum preferisce < >. Non domandatemi perché.

In generale si procede così. L'istruzione cruciale ha la forma

```
IF questo THEN quello
```

dove "questo" è la condizione e "quello" l'istruzione. (Nella precedente linea 20, "questo" era "n = 2 * INT (n/2)" e "quello" era "PRINT "Pari" ").

L'espressione "questo" deve essere qualcosa che il computer sappia riconoscere come vera oppure falsa. (IF STOP THEN... non è estremamente significativa, mentre IF n = 1981 THEN... sì). Se "questo" è vero, il computer continua facendo "quello"; se "questo" è falso, *va alla successiva linea di programma senza eseguire "quello"*.

Seguono due tipi particolarmente comuni di comandi condizionali.

Salti condizionali

I salti condizionali hanno la forma

```
10  IF questo THEN GO TO n
```

dove n è un numero di linea. Istruzioni come questa si possono usare per modificare il corso della computazione indirizzandolo verso un'altra parte del programma.

Per esempio, se si vuol calcolare la radice quadrata di un numero, è importante ricordare che per numeri negativi non esiste. Potete evitare messaggi d'errore facendo qualcosa del genere:

```
10  INPUT n
20  IF n < 0 THEN GO TO 50
30  PRINT n, SQR n
40  STOP
50  PRINT "Radice quadrata non definita"
```

Notate lo STOP alla linea 40. Perché? Eliminatelo per vedere che cosa succede!

Allo stesso modo, quando usate PRINT AT a, b, potete cautelarvi contro valori di a e b non stampabili facendo uso di:

```
 100  IF a < 0 THEN GO TO 1000
 110  IF a > 21 THEN GO TO 1000
 120  IF b < 0 THEN GO TO 1000
 130  IF b > 31 THEN GO TO 1000
 150  PRINT AT a, b; "*"
 160  STOP
1000  qualsiasi cosa riteniate utile...
```

Qui supponiamo che a e b siano definiti in una parte precedente del programma.
Aggiungete all'inizio

```
10   INPUT a
20   INPUT b
```

e chiedete alla macchina di stampare alla posizione 999, —37, impostando a = 999 e b = —37.
Fate sì che la linea 1000 sia

```
1000   PRINT "Non sono così stupido!"
```

Se pensate che le linee 100–130 siano un esempio di programmazione poco raffinato... ebbene avete ragione. Leggete il paragrafo *Logica* poco più avanti.

Assegnazioni condizionali

Le assegnazioni condizionali hanno la forma

IF questo THEN LET qualcosa

Per esempio, un'alternativa al primo dei programmi precedenti è:

```
10   INPUT n
20   LET a$ = "Dispari"
30   IF n = 2 * INT (n/2) THEN LET a$ = "Pari"
40   PRINT a$
```

Qui a$ ha il segno $ di dollaro, dal momento che non è un numero, ma una *stringa* di caratteri (vedi Capitolo 17).
Che cosa è questo programma?

```
10   LET s = INT (2 * RND)
20   IF s = 0 THEN LET a$ = "Testa"
30   IF s = 1 THEN LET a$ = "Croce"
40   PRINT a$
```

Ogni volta che state scrivendo un programma, e quello che volete fare dipende dal verificarsi o meno di certe condizioni, iniziate a pensare in termini di IF... THEN...

Logica

La logica: grosso argomento, interminabilmente esaminato in dotti trattati... ma che per la maggior parte a noi non serve conoscere. Lo Spectrum sa fare logica, probabilmente meglio di voi o di me. In particolare, è in grado di combinare istruzioni che si presentano nella posizione "questo" di un IF *questo* THEN *quello*, usando gli operatori logici AND, OR e NOT.

Regole fondamentali: p AND q è vero solo quando p è vero e q è vero

p OR è vero quando è vero p, oppure q, oppure *entrambi*

NOT p è vero solo quando p è falso.

Lo Spectrum calcola i NOT prima degli AND e gli AND prima degli OR. Svolge praticamente tutti gli altri tipi di operazione prima di eseguire una qualsiasi istruzione logica. Ne risulta che non vi serviranno spesso la parentesi per chiarire l'ordine di esecuzione.

Per esempio, possiamo migliorare le linee 100 – 130 precedenti, trasformandole nell'unica linea:

```
100  IF a < 0 OR a > 21 OR b < 0 OR b > 31 THEN GO TO 1000
```

Esiste ogni sorta di meravigliosi modi per utilizzare la logica dello Spectrum, ma sono di difficile spiegazione e non ho molto spazio, così con riluttanza mi fermerò qui. Il Capitolo 22 del *Manuale* Sinclair vi introdurrà all'argomento, senza peraltro esaurirlo.

Linee multi-istruzione

Lo Spectrum permette di inserire più di un comando su una sola linea. È solamente necessario separare i vari comandi con i due punti ":" Così il programma al Capitolo 1 sulle basi del BASIC si *potrebbe* scrivere (per esempio):

```
10  PRINT "Raddoppio": INPUT x : LET y = x + x
20  PRINT x, y: STOP
```

E in effetti, se aveste voluto, si sarebbe potuto scrivere addirittura tutto su una sola linea.

Questa tecnica si può usare per risparmiare un poco di spazio (riducendo i numeri di linea) oppure per rendere un programma di più agevole comprensione. La principale difficoltà risiede nel fatto che con GO TO ci si può riferire solo all'inizio di una linea contenente più istruzioni. Nel complesso ho evitato istruzioni multiple — è generalmente più facile seguire lo svolgimento del programma se non vengono utilizzate — ma c'è un'occasione in cui possono essere davvero molto utili. Ho già usato questo trucco in Debugging I:

```
34  IF d = 0 THEN PRINT "Non eseguibile": GO TO 30
```

cosa che permette alla macchina di eseguire *due* azioni al verificarsi di una determinata condizione, evitando l'uso di numerosi GO TO.

La questione centrale da ricordare è che dopo un THEN *tutti* i comandi di quella linea sono condizionati dall'IF che regge quel THEN. In altre parole un'istruzione

```
IF questo THEN quello: quell'altro: qualcosa ancora
```

porterà all'esecuzione di *o di tutti e tre i comandi* ''quello, quell'altro, qualcosa ancora'' (se l'IF è vero) *o di nessuno* (se la condizione è falsa).

Tutto ciò è molto utile, ma può anche diventare un trabocchetto. È molto facile scrivere

```
10  IF x = 0 THEN GO TO 20: IF x = 1 THEN GO TO 1000
20  PRINT "x vale zero"
```

nella convinzione che la macchina salterà a 20 se x = 0 e a 1000 se x = 1. Non è vero, affatto. Se x = 1 prenderà la linea 10 con il significato:

IF x = 0 THEN...
...GO TO 20 *e* IF x = 1 THEN GO TO 1000

ma, dato che x vale 1, dunque è diverso da 0, dunque la condizione è falsa, THEN...

...ignora il resto della linea e continua con la successiva.

Che è la linea 20, dove *non* volevamo che andasse!

Comunque, sostituendo la linea 10 con

```
10  IF x = 0 THEN GO TO 20
15  IF x = 1 THEN GO TO 1000
```

tutto funzionerà come vi aspettereste.

Morale: *le linee con più istruzioni hanno la massima utilità nei comandi IF/THEN, dove manifestano anche la massima pericolosità.*

10. Disegnare

Una delle ragioni principali per comperare uno Spectrum: la sua splendida grafica

Come molti dei più recenti piccoli calcolatori, lo Spectrum è equipaggiato con alcuni strumenti programmativi sofisticati per poter disegnare. È dotato di quelle che sono note come grafica ad alta risoluzione e a bassa risoluzione. Queste sono espressioni utili per bloccare la conversazione alle feste, ma quello che in realtà vogliono dire è che potete disegnare con una matita (alta risoluzione) oppure con un grosso pennello (bassa risoluzione).

In questo capitolo tratterò solo la generazione di grafica ad alta risoluzione. Quando siete in questo modo, lo schermo si considera suddiviso in un gran numero di quadratini detti pixel, 256 dei quali lungo la direzione orizzontale e 176 in quella verticale. Ci sono quindi 256 × 176 = 45056 pixel in totale.

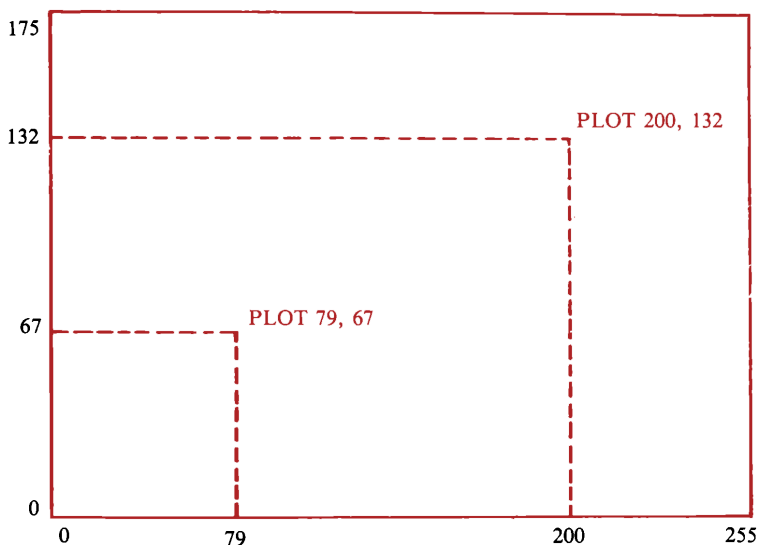



Figura 10.1.

La Figura 10.1 indica come ci si riferisca a questi. La prima colonna è marcata 0 e l'ultima 255. La riga più in basso è la numero 0 e

quella più in alto la 175. Scrivendo, per esempio:

```
50  PLOT 79, 67
```

si annerisce un punto sullo schermo e, come potete vedere dal disegno, il primo valore dopo PLOT indica il numero della colonna, mentre il secondo quello della riga. La parola-chiave PLOT dice al sistema che volete ragionare in termini di alta risoluzione; semplicemente PLOT non si può usare per un lavoro a bassa risoluzione. Esistono altre due parole-chiave utilizzate in alta risoluzione: DRAW e CIRCLE.

Vediamo per prima DRAW. Nella sua accezione più semplice, DRAW serve per tracciare una linea retta. Il punto di partenza della linea è implicito: è la posizione della "matita" in quell'istante. I due valori che seguono DRAW danno il numero delle colonne e il numero delle righe di cui spostarsi per raggiungere il punto terminale della linea. Così

```
10  PLOT 30, 30
20  DRAW 40, 80
```

produrrebbe la linea mostrata in Figura 10.2.

Aggiungendo la linea di programma

```
30  DRAW 50, —10
```

otteniamo il risultato di Figura 10.3.

È quindi molto facile disegnare su comando oggetti come i rettangoli. Supponiamo per prima cosa di inserire i dati del rettangolo, fornendo i valori del suo vertice inferiore sinistro, la sua base e la sua altezza:

```
10  INPUT "Colonna più a sinistra"; cs
20  INPUT "Riga inferiore"; ri
30  INPUT "Altezza"; a
40  INPUT "Base"; b
```

Possiamo iniziare a disegnare dal vertice inferiore sinistro:

```
50  PLOT cs, ri
```

Ora tracciamo la base del rettangolo:

```
60  DRAW b, 0
```

il lato destro:

```
70  DRAW 0, a
```

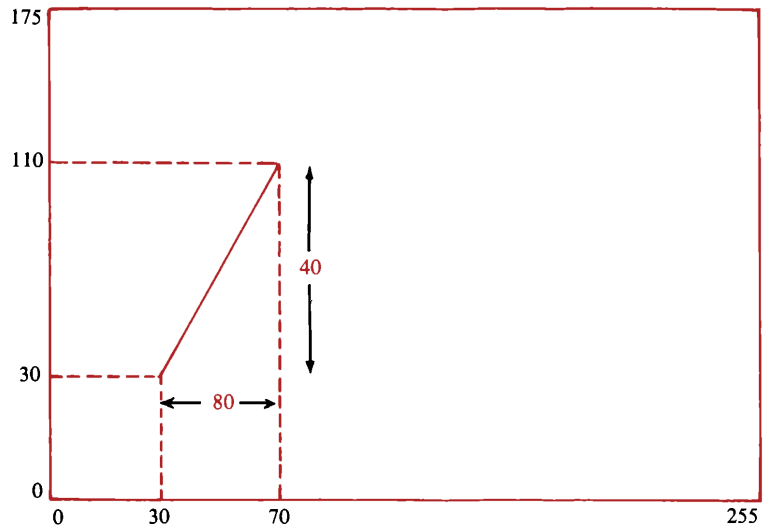



Figura 10.2.

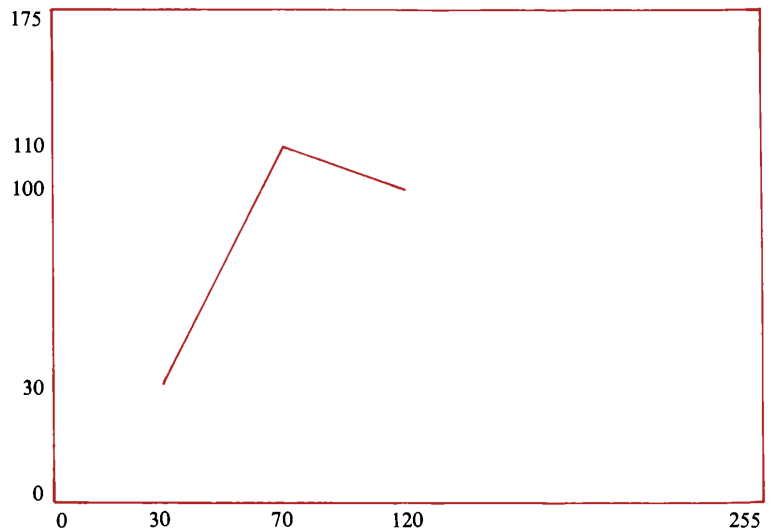



Figura 10.3.

la base superiore:

```
80  DRAW —b, 0
```

e il lato sinistro :

```
90  DRAW 0, —a
```

Se il rettangolo deve essere riempito, le cose sono un attimo più delicate. Ora, abbiamo bisogno di tracciare tutte le linee verticali comprese tra le estremità destra e sinistra. Supponiamo di volerne tracciare solo una: quella della generica colonna c. (Tutto quello che sappiamo di c, per il momento, è che è maggiore di cs, la colonna contenente il lato sinistro, e minore di cs + b, dove si trova il lato de-

stro). Impostiamo quindi le linee di programma:

```
110  PLOT c, ri + 1          [per posizionare la "matita"]
120  DRAW 0, a —1
```

Ora dobbiamo ripetere l'operazione per tutti i valori di c a partire da cs + 1 fino a cs + b — 1. I "+1" e "—1" sono presenti dal momento che non è necessario ridisegnare il perimetro. Ovviamente si può usare un ciclo FOR:

```
100  FOR c = cs + 1 TO cs + b — 1
130  NEXT c
```

A questo punto, per ripetere il procedimento, in modo da poter disegnare rettangoli su tutto lo schermo, non ci rimane che aggiungere:

```
140   GO TO 10
```

Naturalmente, i rettangoli verranno riempiti in ogni caso, siccome le linee da 100 a 130 vengono sempre eseguite. Potremmo invece di volta in volta domandare all'utente se voglia "colorare" il suo rettangolo, in questo modo:

```
48  "Da colorare? (sì/no)"; c$
```

e ignorare le linee relative al "riempimento" se la risposta è "no":

```
95  IF c$ = "no" THEN GO TO 10
```

Tutto questo presuppone che l'utente si stia comportando in modo sensato, e non cercando di disegnare fuori dai limiti dello schermo. Si tratta di un assunto pericoloso. Gli utenti andrebbero considerati, non tanto come sprovveduti che possono fare qualcosa di sbagliato, quanto come individui maliziosi che faranno di tutto per distruggere il vostro programma, se solo date loro mezza possibilità.

Dovremmo perciò provvedere ad inserire alcuni test per assicurarci che quel dato rettangolo si possa effettivamente disegnare.

Per prima cosa, la colonna più a sinistra appartiene allo schermo?

```
15  IF cs < 0 OR cs > 255 THEN PRINT
    "Non disegnabile": GO TO 10
```

Poi, è possibile disegnare la base?

```
25  IF ri < 0 OR ri > 175 THEN PRINT
    "Non disegnabile": GO TO 20
```

L'altezza è negativa?

```
33  IF a < 0 THEN PRINT "Sciocco!": GO TO 30
```

Ci starà la base superiore?

```
36  IF ri + a > 175 THEN PRINT
    "La linea superiore non ci sta": GO TO 30
```

La larghezza è negativa?

```
42  IF b < 0 THEN PRINT "Sciocco!": GO TO 40
```

Ci starà il lato destro?

```
44  IF cs + b > 255 THEN PRINT
    "Il lato destro non ci sta": GO TO 40
```

Dovremmo anche controllare che sia stato risposto solo "sì" o "no" alla domanda della linea 48.

```
49  IF c$ <> "sì" AND c$ <> "no" THEN PRINT
    "Rispondere solo sì o no": GO TO 48
```

Sorge a questo punto una questione interessante di cui dovreste essere informati: per quanto riguarda la macchina, "sì" non è la stessa cosa di "SI". Lasciando il programma come è ora, se l'utente ha inserito il tasto delle maiuscole (cioè scrive tutto in lettere maiuscole) e risponderà "SI" o "NO" alla domanda della linea 48, la macchina infuriata continuerà a replicare

```
Rispondere solo con sì o no
Da colorare? (sì/no)
```

Cercate di modificare la linea 49 in modo che la macchina permetta all'utente di rispondere sia in caratteri maiuscoli che minuscoli.

Girotondo con lo Spectrum

Disegnare cerchi è molto facile: esiste infatti un'apposita parola-chiave, CIRCLE, dopo la quale è necessario specificare la colonna e la riga dove volete porre il centro, e, di seguito, il valore del raggio. Per esempio:

```
20  CIRCLE 50, 70, 30
```

farà tracciare un cerchio il cui centro si trova all'intersezione della colonna 50 e della riga 70, avente raggio 30.

Anche DRAW, comunque, si può utilizzare per disegnare cerchi, o, piuttosto, archi di circonferenza.

Provate:

```
10  PLOT 20, 30
20  DRAW 60, 100
30  DRAW —60, —100, 1
```

Vedrete come la linea 20 tracci una linea retta, proprio come vi sareste aspettati, e come la linea 30 torni al punto di partenza (disegnato da PLOT), ma seguendo un cammino circolare invece di una linea retta. È la terza variabile che segue DRAW a dire al BASIC di disegnare un arco di circonferenza; il suo funzionamento non è semplice, ma, ciononostante, ho intenzione di spiegarlo.

Immaginate la circonferenza completa (la cui parte non disegnata è indicata a tratteggio in Figura 10.4).

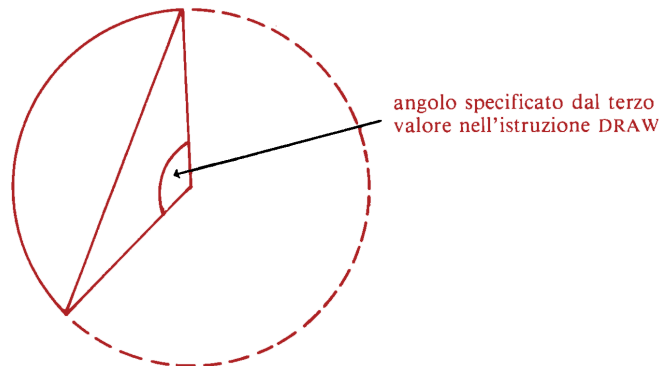



Figura 10.4.

Il terzo valore che segue DRAW indica l'ampiezza dell'angolo compreso tra i due raggi (della circonferenza non tracciata) che definiscono le estremità dell'arco. Dal momento che quest'angolo è ovviamente molto più grande di 1, potreste trovare la cosa sorprendente. La spiegazione, giusto per non semplificare troppo la vita, sta nel fatto che l'angolo è misurato in radianti, e un radiante è poco meno di 60°. Ora, se tutto ciò non è cristallino e voi pensavate che i radianti fossero quei pneumatici che sembrano a terra anche quando sono perfettamente gonfi, non preoccupatevi. In pratica, tutto ciò significa che se scegliete un angolo piccolo (diciamo 0.1), otterrete una linea che non devia di molto dalla retta. Scegliendo angoli maggiori, la natura circolare della linea diventa più pronunciata. Provate:

```
10   PLOT 60, 20
15   FOR a = 0.4 TO PI STEP 0.4
20   DRAW 100, 0
30   DRAW —100, 0, a
50   NEXT a
```

Vedete come la forma si avvicini a un semicerchio, man mano che l'angolo si avvicina a PI (3.14159)?

Ora trasformate la linea 15 in:

```
15   FOR a = 0.4 TO 2 * PI STEP 0.4
```

Otterrete una parte sempre maggiore di un cerchio completo che alla fine non entrerà più nello schermo. Non importa quanto sia piccola la retta originale, prima o poi il cerchio non entrerà più nello schermo. Il motivo è che 2 * PI radianti sono 360° — un cerchio

completo! Dal momento che questo cerchio include la linea retta originale — un segmento diritto — deve avere un raggio infinito! Morale: non lasciate che l'angolo diventi troppo grande (circa 5 vi dà la maggior parte di un cerchio) e anche così, fate attenzione: è facile andare fuori dallo schermo a tutta velocità e non è facile eseguire dei test preliminari per stabilire se la cosa stia per accadere.

Riempire i buchi

Abbiamo visto come riempire un rettangolo, ma riempire un cerchio o un settore circolare sembra un proposito più arduo. Il motivo è che è più difficile stabilire dove cominciare e finire la DRAW che esegue la colorazione, dal momento che questi valori non sono più fissi come per il rettangolo. E questo è il problema: serve un metodo per scoprire dove si trovano le estremità della figura, per ogni riga che vogliamo "colorare". (Tanto per cambiare, coloreremo le righe invece delle colonne).

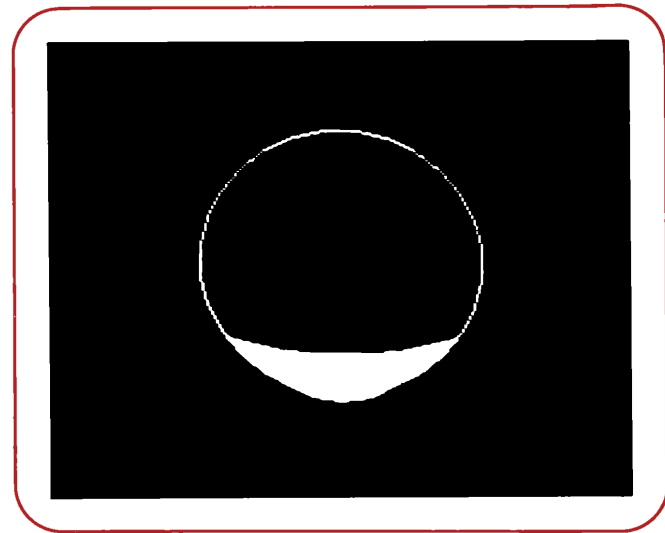
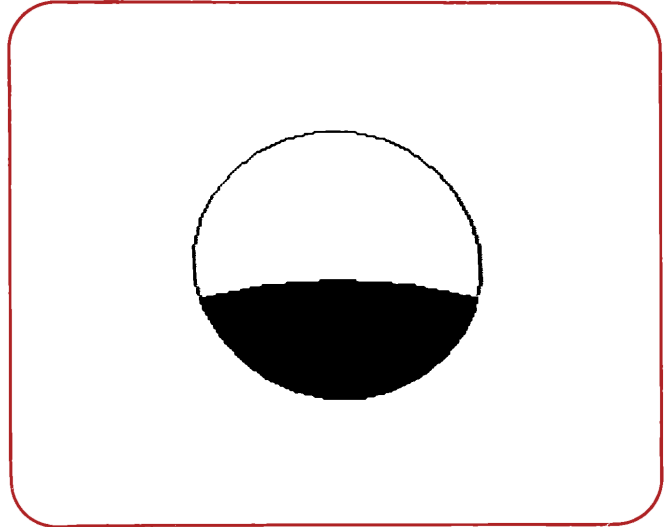
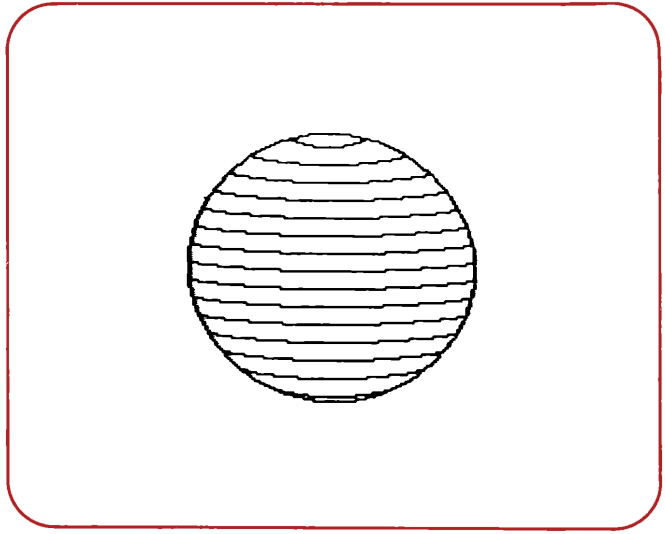
Fortunatamente, lo Spectrum ci dà modo di scoprire se un particolare pixel sia colorato, per mezzo della funzione detta POINT. Se

```
200  LET g = POINT (20, 30)
```

g sarà posta a 1 se il pixel a 20, 30 è colorato; a 0 in caso contrario. Il problema si spezza come segue:

1. Stabilire uno spazio rettangolare intorno alla figura da colorare, in cui deve aver luogo la ricerca degli estremi.
2. Per ogni riga:
   (a) Cercare da sinistra finché si colpisce la figura. Prendere nota di dove ciò accade.
   (b) Cercare da destra finché si colpisce la figura. Prendere nota di dove ciò accade.
   (c) Tracciare una linea tra i due punti trovati in (a) e (b).

Ecco il programma risultante:

```
600  INPUT "Colonne cornice"; cs, cd    [le colonne sinistra e
                                         destra del rettangolo
                                         d'incorniciatura]
610  INPUT "Righe cornice"; ri, rs      [le linee inferiore e
                                         superiore del rettangolo
                                         d'incorniciatura]
620  FOR r = ri TO rs
630  FOR c = cs TO cd                   [ricerca da sinistra a
                                         destra...
640  IF POINT(c,r) = 1 THEN GO TO 660   ...dell'estremità]
650  NEXT c
655  GO TO 730                          [se il programma arriva
                                         qui, non c'è una
                                         estremità in questa riga,
                                         perciò va alla prossima]
```

```
660  LET c1 = c                      [c1 corrisponde alla
                                    colonna più a sinistra
                                    della figura]
670  FOR c = cd TO cs STEP —1       [ricerca da destra
                                    a sinistra]
680  IF POINT(c,r) = 1 THEN GOTO 700
690  NEXT c
700  LET c2 = c                      [c2 è la colonna più a
                                    destra della figura]
710  PLOT c1, r                     [da qui...
720  DRAW c2 — c1, 0                ...traccia la linea]
730  NEXT r
```

Naturalmente dovete far precedere una routine per disegnare una figura chiusa, come quella per disegnare un cerchio all'inizio del paragrafo precedente, per poter vedere qualche cosa.

Ci sono alcune semplici modifiche che rendono questo programma piuttosto potente.

Per prima cosa manipolate la linea 620:

```
620  FOR r = ri TO rs STEP s
```

e scrivete una linea che permetta all'utente d'inserire un valore di s a piacimento (in un luogo conveniente prima di 620). Se s viene posto a 1, l'effetto è uguale al precedente, ma con s = 2, viene tracciata solo una riga su due, ottenendo un effetto di ombreggiatura invece che di annerimento. Con valori di s più grandi, l'ombreggiatura si fa più rada, naturalmente.

Ora modificate la linea 720:

```
720  DRAW c2 — c1, 0, a
```

e inserite "a" con un INPUT in luogo conveniente. Provate a ombreggiare un cerchio con un valore piuttosto piccolo di "a" (per esempio 0.5).

Visto l'effetto tridimensionale?

Ora scegliete il rettangolo d'incorniciatura in modo che la metà superiore del cerchio ne rimanga al disopra. Usate lo stesso valore di a, ma riportate s a 1. Abbiamo ottenuto una luna nell'oscurità. (Usate inchiostro bianco su carta nera, per avere l'effetto migliore). Eccetera... Quando scrissi questo programma per la prima volta, mi divertii molto a disegnare penne d'oca entro calamai. Cosette...

La gomma elettronica del BASIC

Cancellare delle figure è divertente quasi quanto disegnarle. Allo scopo si utilizzano ancora i comandi PLOT, DRAW e CIRCLE, ma è necessario dire al BASIC che avete in mano la gomma, non la matita. Si fa questo con il comando: OVER 1.

Provate:

```
50  CIRCLE 120, 85, 60
60  OVER 1
70  CIRCLE 120, 85, 60
```

e vedrete disegnare e poi cancellare il cerchio. Ora sostituite la linea 70 con GO TO 50 e fate nuovamente girare il programma. La circonferenza viene disegnata, poi cancellata, quindi ridisegnata, e così via!

Va bene, finora vi ho raccontato frottole. In realtà nello stato OVER 1 il calcolatore cancella, se c'è qualcosa da cancellare, ma altrimenti disegna. Perciò la prima volta che viene eseguita la linea 50, viene tracciata la circonferenza, la volta successiva viene cancellata (perché è là, da cancellare), la volta successiva viene ridisegnata (dato che non c'è niente da cancellare), e così via.

Provate ora:

```
10  OVER 0                          [per escludere l'azione di
                                     cancellazione]
20  CIRCLE 40, 40, 30
30  CIRCLE 60, 40, 30
40  OVER 1                          [attiva la cancellazione]
50  CIRCLE 100, 100, 30
60  CIRCLE 120, 100, 30
```

Vedete l'effetto? Con la cancellazione disinserita i primi due cerchi vengono disegnati e si sovrappongono proprio come vi aspettereste. Con la cancellazione inserita, anche l'altra coppia di cerchi si sovrappone, ma il secondo cancella i punti del primo dove si ha intersezione.

Il funzionamento di CIRCLE (o DRAW o PLOT) sotto la condizione OVER 1 è dunque: "Se c'è qualcosa da cancellare, cancellalo; altrimenti disegna la forma indicata". Nel programma precedente, l'azione di OVER 0 alla linea 10 dura finché non viene annullato da OVER 1, in linea 40. È possibile far sì che un comando OVER valga per una sola istruzione nel modo seguente:

```
70  CIRCLE OVER 0; 130, 100, 30
```

Il nuovo cerchio si sovrappone al primo senza cancellare le intersezioni, ma qualsiasi istruzione che segua non contenente a sua volta OVER 0 (provate per esempio 90 PLOT 0, 0: DRAW 150, 148) manterrà l'OVER 1 di linea 40 e quindi cancellerà i punti d'intersezione, benché, laddove il punto d'intersezione è un singolo pixel, l'effetto non sia molto appariscente. Che effetto avrebbe l'eliminazione di OVER 0 dalla linea 70? Provate!

Tramandate la vostra arte ai posteri

Potete usare le tecniche che abbiamo discusso per disegnare dei quadri fantastici, e c'è un "Programma preconfezionato" per rendere la cosa piuttosto semplice (fa uso delle procedure per colorare e ombreggiare descritte in questo paragrafo).

Sarebbe comodo salvare i risultati su nastro magnetico. Per esempio potremmo disegnare una bella veduta della superficie lunare che ci piacerebbe riutilizzare in un programma di "allunaggio"; oppure una serie di disegni che rappresentino ciascuno una buca di un percorso di golf. Un programma "Golf" potrebbe caricarli in sequenza man mano che si giocano le varie buche.

Lo Spectrum rende estremamente facile questo processo. Memorizziamo e ricarichiamo il contenuto dello schermo quasi esattamente nello stesso modo in cui lo facciamo per i programmi. La sola differenza consiste nello scrivere "SCREEN$" dopo la normale istruzione SAVE o LOAD per indicare che è lo schermo che scarichiamo (o rileggiamo) sul nastro, e non un programma. Per esempio

```
SAVE "BUCA GOLF3" SCREEN$
```

dice: "salva il contenuto dello schermo sotto forma di un file[1] su nastro chiamato BUCA GOLF3", e per andare a riprenderlo:

```
LOAD "BUCA GOLF3" SCREEN$
```

---

(1) Un file è un flusso seriale di dati o il dispositivo destinato ad accoglierlo (o generarlo). Oltre al nastro, esempi tipici di file si hanno con la tastiera, alcuni terminali periferici, i modem ecc. [NdT].

11. Debugging II

Carta e matita servono ancora...

In precedenza vi ho lasciato con il programma di calcolo della media che è riprodotto di seguito (così non dovete voltare troppe pagine) e vi ho suggerito di provare a correggerlo.

```
10  LET s = 0
20  LET c = 0
30  INPUT n
40  IF n < 0 THEN GO TO 100
50  LET c = c + i
60  LET z = s + n
70  GO TO 30
100 PRINT "La media è □"; s/c
```

Prendiamo ora in considerazione un approccio passo-passo al problema. Ovviamente, la prima cosa da fare è provare a vedere se funziona così com'è, quindi lo introduciamo nello Spectrum e lo mandiamo in esecuzione con alcuni semplici insiemi di dati. Supponiamo di provare:

```
 3
 7
 5
—1     [ricordate che questo valore serve solo come
       terminatore o tappo]
```

La risposta *dovrebbe* essere 5.

In effetti, quando il programma viene fatto girare, si ottiene il responso d'errore 2 Variable not found, 50: 1. (Variabile non trovata). Hmmm. Significa che nella linea 50 è stata usata una variabile precedentemente non definita. Dal listato si può vedere che si tratta

di “i”. Diamo perciò un valore a i:

```
5  LET i = 0
```

Questo metterà a posto le cose, così il programma procede oltre la linea 50. Aggiungendo la linea 5 e ridando RUN, il messaggio

La media è

viene correttamente stampato, ma dopo questo appare il messaggio d’errore: 6 Number too big, 100: 1 [Numero troppo grande].

Abbiamo già incontrato in precedenza questo messaggio: probabilmente ricordate che “6” indica la macchina ha cercato di eseguire un’operazione aritmetica che conduce a un risultato maggiore di quello che può contenere, e che “100” indica il numero di linea dove è sorto il problema. Si potrebbe scommettere che il programma ha cercato di dividere per zero, come nel caso dell’ultima volta in cui si è presentato questo responso d’errore. Prima d’iniziare il lavoro di investigazione vero e proprio, raccogliamo qualche altra prova. Proviamo con un insieme di dati diverso:

```
 2
 1
 4
 6
—1
```

Si ha esattamente la stessa serie di messaggi. Una ipotesi di lavoro sembra quindi “Qualsiasi siano i dati inseriti, il programma termina con un responso di overflow aritmetico”. Provate il programma con altri tre o quattro insiemi di dati e vedrete che la nostra ipotesi sembra sempre più valida.

Da dove cominciare a cercare? Un minuto fa ho utilizzato le parole “investigazione” e “prova”; e non era a caso. Ogni debug di un programma è un lavoro d’investigazione, e qualsiasi investigatore vi dirà che dovete cercare di pensare come i criminali, per avere successo, o, se preferite “ci vuole un ladro per prendere un ladro”. Il criminale di questo racconto è lo Spectrum, così il nostro problema è di cercare di pensare come “lui”. La prima cosa da fare è di rallentare i vostri processi mentali. Sorpresi, no? Avevate l’impressione che i computer fossero piuttosto rapidi. Ebbene, lo sono, ma il modo in cui “pensano” è solitamente alquanto laborioso. Quindi proviamo a elaborare un modello della memoria del calcolatore, o almeno di quella parte di essa che ha importanza per il problema in questione. Nel nostro esempio sono stati fissati cinque elementi di memoria a cui sono stati assegnati i nomi “s”, “c”, “n”, “i” e “z”. Un utile modello quindi sarà semplicemente una tabella nella quale si possa indicare come variano i contenuti di questi elementi nel corso dell’esecuzione del programma. Può inoltre essere d’aiuto un’indicazione del numero di linea in cui ha luogo una decisione, benché ciò non sia essenziale.

La tabella potrebbe quindi avere il seguente aspetto:

Numero di linea	s	c	n	i	z	Decisione

Ho aggiunto una colonna extra indicata con "Decisione" della cui funzione dirò fra breve. Ora costruiamo la tabella considerando in sequenza l'azione di ogni istruzione. Dobbiamo definire un insieme di dati da trattare; scegliamo:

2
1
4
6
—1

Bene, le prime istruzioni eseguite sono alle linee 5 e 10, e pongono semplicemente a zero gli elementi di memoria detti i e s:

Numero di linea	s	c	n	i	z	Decisione
5				0		
10	0					

Dopo aver eseguito la linea 20 abbiamo:

Numero di linea	s	c	n	i	z	Decisione
5				0		
10	0					
20		0				

La linea 30 prende il primo valore dall'insieme dei dati e lo inserisce in n, così:

Numero di linea	s	c	n	i	z	Decisione
5				0		
10	0					
20		0				
30			2			

La linea 40 è una istruzione IF e perciò attua semplicemente un test per stabilire se n è negativo. Non lo è (il valore corrente è 2), perciò il test è falso, cosa che indicheremo con una "x" nella colonna "Decisione" e non ha luogo la diramazione verso la linea 100. Quando il test è vero e il programma si dirama effettivamente, lo indicheremo con una "√" nella colonna "decisione". Ora abbiamo:

Numero di linea	s	c	n	i	z	Decisione
5				0		
10	0					
20		0				
30			2			
40						x

La linea 50 domanda alla macchina di prendere il contenuto di i, sommarlo al contenuto di c e di riporre il risultato in c. La cosa dovrebbe insospetterci, perché sappiamo che i non era stato definito in origine. Può darsi che aggiungere la linea 5 non sia stata un'idea così buona. In ogni caso, continuiamo con l'esecuzione simulata.

Numero di linea	s	c	n	i	z	Decisione
5				0		
10	0					
20		0				
30			2			
40						x
50		0				

Naturalmente sommare zero a zero non ha avuto alcun effetto utile e questo dovrebbe renderci ancora più guardinghi, perciò, ricordando che stiamo cercando errori tipografici, potremmo sospettare che i non fosse affatto i, ma 1, una svista piuttosto comune, dopo tutto.

Ricominciamo tutto daccapo con una nuova tabella, lavorando sul nuovo assunto che la linea 5 non è necessaria e che alla linea 50 si legga: 50 LET c = c + 1.

Numero di linea	s	c	n	z	Decisione
10	0				
20		0			
30			2		
40					x
50		1			
60				2	
70					√
30			1		
40					x
50		2			
60				1	
70					√
30			4		
40					x
50		3			
60				4	
70					√
30			6		
40					x
50		4			
60				6	
70					√
30			-1		
40					√
100					

Quando il programma raggiunge la linea 100, stamperà il messaggio "La media è" seguito dal valore di s/c, che è zero!

Il programma quindi non funziona ancora, ma la cosa interessante è che adesso non c'è segnalazione d'errore da parte dello Spectrum; abbiamo introdotto un nuovo tipo di errore — un errore logico. Lo Spectrum può ben eseguire le operazioni che gli abbiamo richiesto; è solo che, così facendo, ottiene la risposta sbagliata.

Diamo uno sguardo alla tabella che abbiamo ottenuto. Ora è piuttosto chiaro quale sia la funzione di c. Tutte le volte che si inserisce un valore in n, c viene incrementato di uno, in modo che quando avviene la diramazione per la linea 100, c contiene il numero totale dei valori inseriti, in questo caso 4. Ma a s non è successo niente del tutto, dopo che è stato posto a zero, mentre z contiene solo il valore di n, ma un istante più tardi. Forse s e z in realtà dovrebbero essere la stessa cosa, e dal momento che s è stato menzionato per primo alla linea 10, assumiamo che z sia un errore di stampa. Questo significa che la linea 60 va intesa:

```
60  LET s = s + n
```

e la tabella diventa:

Numero di linea	s	c	n	Decisione
10	0			
20		0		
30			2	
40				x
50		1		
60	2			
70				√
30			1	
40				x
50		2		
60	3			
70				√
30			4	
40				x
50		3		
60	7			
70				√
30			6	
40				x
50		4		
60	13			
70				√
30			-1	
40				√
100				

Ora ci viene stampato

La media è 3.25

che è il risultato esatto!

Il procedimento che ho descritto è detto ''*dry running*'' (''esecuzione simulata'') del programma, ed è un metodo comune di ricerca degli errori. Naturalmente non sempre è necessario introdurre in una tabella di esecuzione tutti i particolari che ho indicato (per esempio, in questo caso i numeri di linea non sono stati molto utili) e spesso non è nemmeno necessario completare la tabella, ma penso che riusciate a capire che è un buon metodo per obbligarvi a pensare nel modo costrittivo in cui pensa la macchina, e allo stesso tempo vi dà una chiara visione di come funziona il programma.

Ora potreste obiettare: ''Tutto questo è molto giusto, ma chi ha intenzione di commettere errori tipografici di questo genere, nel battere il programma?''.

La risposta è che succede sempre, per svariate ragioni. Primo, se copiate un programma da una rivista, c'è la possibilità che l'errore sia presente sulla carta stampata. Secondo, in un programma lungo, è facile che commettiate un errore di interpretazione: I al posto di 1 (se il listato che state copiando ha utilizzato solo lettere maiuscole), la lettera o invece di zero, 2 al posto di z, eccetera. Terzo, anche quando siete voi stessi a scrivere un programma, potete commettere un errore equivalente a una svista tipografica.

Supponiamo, per esempio, che all'inizio del programma definiate una variabile b3. Scrivete il programma nel corso di più giorni, rappezzando un pezzo qua, modificando una linea là. Alla fine di tutto questo tempo dovete scrivere ancora qualche linea usando questa variabile, che ricordate benissimo chiamarsi b2, così non vi disturbate a controllare.

Pensate che non possa succedere? Aspettate finché non avrete un esperienza di programmazione di qualche mese.

12. Grafica

Lo Spectrum può disegnare oggetti e farli muovere. L'operatore può controllare il movimento tramite la tastiera. Caratteristiche utili se volete scrivere dei programmi di giochi

L' istruzione PRINT

Il Capitolo 10 descriveva un modo di utilizzare il computer per disegnare, ma allora stavamo pensando a oggetti piuttosto accademici, come rettangoli e circonferenze. Usando l'istruzione PRINT, è possibile disegnare oggetti dotati di maggiori attrattive visuali. In questa sede l'istruzione fondamentale è PRINT x$, dove x$ è un carattere, o una stringa di caratteri. Si spiega praticamente da sola. Compiendo esperimenti con PRINT, tuttavia, scoprirete presto che questa istruzione da sola vi dà un controllo limitato su *dove* stampare un singolo carattere: la macchina inizia qualsiasi PRINT sulla sinistra dello schermo, alla prima linea libera.

PRINT AT x, y risolve questo problema. Similmente al caso di PLOT, lo schermo va considerato diviso in quadrati, etichettati ognuno da due coordinate x e y.

Il primo numero, x, è il numero della riga sullo schermo; va dall'alto verso il basso, da 0 a 21. Il secondo numero, y, è il numero della colonna, che fornisce la distanza lungo una linea (cioè orizzontalmente), e va da 0 a 31. La figura 12.1 mostra il sistema di coordinate in maggiore dettaglio.

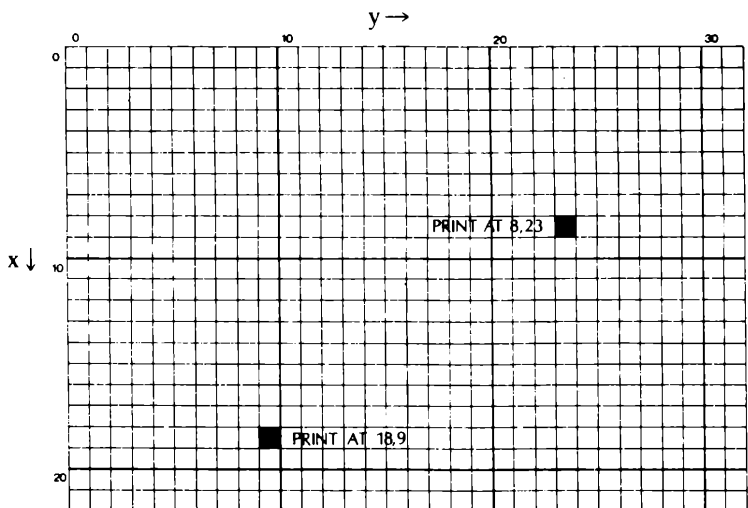



Figura 12.1.

Supponiamo che vogliate stampare il carattere grafico ■ nel centro dello schermo. Non è possibile arrivare al centro esatto; ma il quadrato alla linea 11 e colonna 15 gli è sufficientemente vicino. Dovreste scrivere:

```
10  PRINT AT 11, 15; "■"
```

Notate il punto e virgola (;) che serve per dire al computer di eseguire ciò a cui si riferisce come *due* istruzioni insieme: vai a 11, 15; PRINT qualcosa.

Collegando più caratteri grafici, potete ottenere effetti ancora più interessanti. Il modo più diretto per farlo, usare un gran numero di PRINT AT, consuma memoria preziosa nello stesso modo in cui una Cadillac consuma benzina; ma per il momento non preoccupiamoci dell'efficienza: interessa solo il principio. Provate:

```
10  PRINT AT 10, 10; "■■■▀▀▀▀"
20  PRINT AT 11, 8; "■■■■■■■■"
30  PRINT AT 12, 8; "▐OOOOO▌"
```

Dovrebbe ricordarvi qualcosa di natura militare (se no, rimproverate il mio povero disegno).

Quando stampate disegni grafici come il precedente, un approccio utile consiste nell'eseguire il disegno su carta quadrettata, e numerare linee e colonne; questo facilita la lettura delle istruzioni necessarie. Il programma precedente è stato ottenuto con questo procedimento dalla Figura 12.2

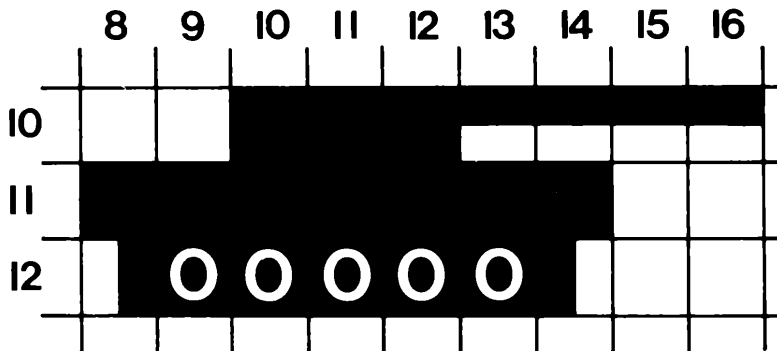



Figura 12.2.

L'insieme dei caratteri dello Spectrum ha sedici caratteri grafici speciali (numeri 128 - 143, vedi *Manuale* pag. 144). Tuttavia, come nell'esempio precedente, potete fare buon uso anche di altri caratteri. I caratteri in campo inverso (bianco su nero) risultano particolarmente utili allo scopo.

Cambiamento di posizione

Dopo aver elaborato una stampa in una data posizione dello schermo, non è difficile modificare il programma in modo da poter avere la stessa figura in una posizione qualsiasi dello schermo. Il carro ar-

mato (sì, è proprio questo che doveva essere) di cui sopra si trova alla linea 12, con l'estremo sinistro nella colonna 8. Se vogliamo ridisegnarlo alla linea a, con l'estremità sinistra nella colonna b, dobbiamo solo riassegnare i numeri alle linee e alle colonne della figura (vedi Figura 12.3)

È possibile scrivere il nuovo programma:

```
10  PRINT AT a — 2, b + 2; "■■■▀▀▀▀"
20  PRINT AT a — 1, b; "■■■■■■■"
30  PRINT AT a, b; "▐ooooo▌"
```

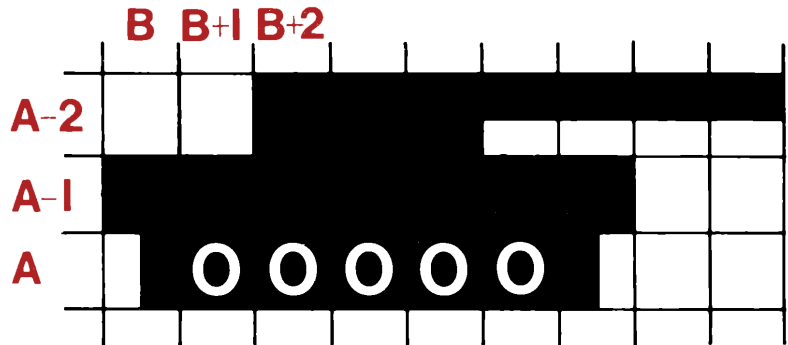



Figura 12.3.

Naturalmente non funzionerà, se non fornite i valori di a e b.

Non funzionerà anche nel caso in cui trovi valori di a e b che facciano uscire la figura dal bordo dello schermo. Osservando il lato sinistro, ciò significa che b deve essere maggiore o uguale a 0; sulla destra occorre che b + 9 sia minore o uguale a 31, cioè b < = 22. In modo analogo a deve essere maggiore o uguale a 2 e minore o uguale a 21.

Il grosso vantaggio di pensare in termini generali, come in questo caso, è che possiamo disegnare i nostri carri armati dovunque sullo schermo, solo specificando a e b. Seguono alcuni esempi: aggiungeteli alle precedenti linee 10-30 (i numeri di linea verranno ordinati automaticamente da parte dello Spectrum) e provate a far girare i programmi così ottenuti.

(a)
```
 1  LET b = 7
 2  FOR a = 2 TO 21
40  NEXT a
```

(b)
```
 1  LET a = 15
 2  FOR b = 0 to 22
40  NEXT b
```

(c)
```
 1  LET a = 2 + 15 * RND
 2  LET b = 20 * RND
40  GO TO 1
```

(Imponete BREAK a quest'ultimo, per fermarlo).

Quando conoscerete i sottoprogrammi, riuscirete a trovare ogni sorta di utilizzazione alla capacità di disegnare una forma data in una posizione arbitraria. Il paragrafo successivo descrive una circostanza molto comune a questo riguardo.

Grafica animata

Supponiamo di desiderare che il carro armato attraversi lo schermo da sinistra a destra. Il programma (b) precedente ottiene quasi questo risultato, senonché lascia dietro di sé una traccia composta di "code" di carri armati. È *possibile* sbarazzarsene stampandovi sopra degli spazi bianchi; ma il modo più elegante è usare un "bordo invisibile" che automaticamente esegue questa cancellazione. Modificate così le linee 10-30:

```
 1  LET a = 15
 2  FOR b = 0 TO 21
10  PRINT AT a — 2, b + 2; "□■■■▀▀▀▀"
20  PRINT AT a — 1, b; "□■■■■■■■"
30  PRINT AT a, b; "□▐◙◙◙◙◙▌"
40  NEXT b
```

Notate gli spazi vuoti aggiunti all'inizio di ogni linea grafica. (Per evitare di uscire dal bordo, alla linea 2 abbiamo fatto andare b solo fino a 21).

Il bordo di spazi non risulta visibile sullo schermo; ma, per il modo in cui vengono stampati dal computer, questi spazi bianchi finiscono sopra quella inopportuna "traccia", cancellandola. (Un vecchio adagio dice che le volpi cancellano le loro impronte con la coda: il nostro carro armato fa lo stesso, ma la sua coda è invisibile).

Volendo poter invertire la direzione del carro armato, abbiamo bisogno di un bordo invisibile anche sulla destra. Per un movimento verso l'alto, ci serve un altro bordo sotto il carro; per un movimento verso il basso, ne occorre uno sopra. Per poter andare in tutte le direzioni (variando opportunamente a e b) poniamo un bordino tutt'intorno alle estremità del carro armato, ciò che significa aggiungere ulteriori linee di spazi bianchi alle righe a — 3 e a + 1.

Progetto

Scrivete un programma per far muovere il carro armato lungo il perimetro di un quadrato, di lato, sullo schermo. Ponete una cornice invisibile tutt'intorno; calcolate come debbano variare a e b; stendete il programma.

Controllo del movimento con la tastiera

Ora che siamo in grado di disegnare oggetti che si muovono, è possibile usare la tastiera per dirigerne il movimento.

Un modo piuttosto goffo per arrivare al risultato è di scrivere il programma sotto forma di ciclo, e di usare le INPUT per inserire qualcosa dalla tastiera. Questo metodo blocca tutto, figura in movimento e tutto il resto, finché non vengono premuti i tasti.

Meglio l'istruzione INKEY$. Una linea di programma come

```
10  LET c$ = INKEY$
```

ordina alla macchina di guardare quale tasto è premuto, e di assegnare il carattere corrispondente alla variabile di tipo stringa c$.

Per esempio, dirigiamo il carro armato a destra o a sinistra, usando i tasti 5 (per la sinistra) e 8 (destra). Questa scelta è un utile pro memoria, a causa delle frecce stampate su quei tasti; *non* è tuttavia necessario premere CAPS SHIFT ed introdurre effettivamente le frecce. Cosa dobbiamo fare? Diciamo alla macchina di leggere INKEY$ e di regolare la posizione di stampa secondo l'azionamento di 5 o di 8. In questo modo:

```
 1  LET b = 15
 2  LET a = 12
 3  LET c$ = INKEY$
 6  IF c$ = "5" THEN LET b = b — 1
 7  IC c$ = "8" THEN LET b = b + 1
10  PRINT AT a — 2, b + 2; "□■■■▀▀▀▀□"
20  PRINT AT a — 1, b; "□■■■■■■■□"
30  PRINT AT a, b; "□▐◙◙◙◙◙▌□"
40  GO TO 3
```

Notate la cornice invisibile da entrambi i lati. Il problema con questa versione del programma è che, se tenete i tasti premuti troppo a lungo, potete uscire dallo schermo. Studiate una modifica al programma per prevenire questa eventualità, aggiungendo delle linee del tipo IF a < 0 THEN... o qualcosa d'altro.

Se tenete premuto un tasto, questo programma continua a rispondere, e il carro armato continua a muoversi. A volte questo effetto è piuttosto fastidioso; spesso ciò che in realtà si desidera è che il programma risponda solo a *variazioni* degli INKEY$, o almeno a un nuovo azionamento dei tasti.

```
3  IF INKEY$ <> " " THEN GO TO 3
4  IF INKEY$ = " " THEN GO TO 4
5  LET c$ = INKEY$
```

Questo frammento di programma ha l'effetto di bloccare tutto alla linea 3 se state premendo ancora lo stesso tasto. Quando lo lasciate andare il programma va alla linea 4 e vi rimane. Quando premete un nuovo tasto o ripremete lo stesso, riprende ad avanzare!

ATTENZIONE: siate estremamente cauti su ciò che dite alla macchina con INKEY$. Può essere che *desideriate* premere solo tasti numerici, e quindi eseguire VAL INKEY$, che converte quel numero da un carattere in qualche cosa che potete effettivamente usare per calcoli aritmetici[1]. Tuttavia per cominciare il programma dovete

premere ENTER — e a volte lo Spectrum legge *questo* con INKEY$, cerca di convertirlo in un numero e il programma collassa. O peggio, se non ci sono tasti premuti, esegue VAL (stringa vuota). Questa può essere una sciocchezza tale da mettervi in imbarazzo finché non ve ne accorgete. (Potete tutelarvi contro queste possibilità, usando delle opportune istruzioni IF... THEN...)

Uso combinato di PRINT e PLOT

In alcuni programmi potreste avere bisogno di usare insieme sia PRINT che PLOT per delle figure grafiche. È importante ricordare che PRINT AT x, y e PLOT x, y fanno riferimento a sistemi di coordinate molto diversi sullo schermo. Il *Manuale* Sinclair a pag. 144 reca un diagramma che li mostra entrambi. Solitamente il modo più semplice per fare sì che i due tipi di comandi coesistano in modo coerente, è quello di preparare uno schizzo di massima, su carta quadrettata, della zona in cui volete disegnare; segnare entrambi i sistemi di coordinate; fare riferimento a questo schizzo mentre scrivete il vostro programma. È spesso ben speso il tempo passato a pensare alla struttura del programma, *prima* di sedersi alla tastiera. (Per contro, se il vostro primo tentativo fallisce, è spesso più facile fare il debugging tramite esperimenti alla macchina).

PAUSE

Lo Spectrum ha un comando PAUSE che lo pone in attesa per un determinato periodo di tempo. Risulta molto utile per la grafica animata, ad esempio per rallentare qualcosa che altrimenti si muoverebbe troppo rapidamente. Per attendere n secondi scrivete:

```
PAUSE 50 * n
```

---

[1] NdT: premendo il tasto 5, INKEY$ restituisce *la stringa* "5" che *non* è un numero, ma una stringa di lunghezza unitaria (se volete, un carattere) il cui VAL è 5.

13. Caratteri definiti dall'utente

Se il set di caratteri disponibili non vi basta, inventatene voi stessi di nuovi

Lo Spectrum ha 21 caratteri che potete modificare a piacere. Sono i numeri 144-164 del set di caratteri che, inizialmente, corrispondono alle lettere A-U. Per accedervi dalla tastiera, andate nel modo "G" e battete il tasto della lettera corrispondente.

La tecnica per costruire i vostri caratteri grafici personalizzati è basilarmente semplice, ma richiede una spiegazione accurata, poiché è una caratteristica molto elegante, e può trasformare un programma altrimenti banale, in qualcosa di molto speciale.

Il primo passo consiste nel fare uno schizzo del carattere desiderato su una griglia 8 × 8, annerendo quei quadretti che saranno scuri una volta che il carattere sia stampato. Per esempio, la Figura 13.1 mostra un carattere "gatto".

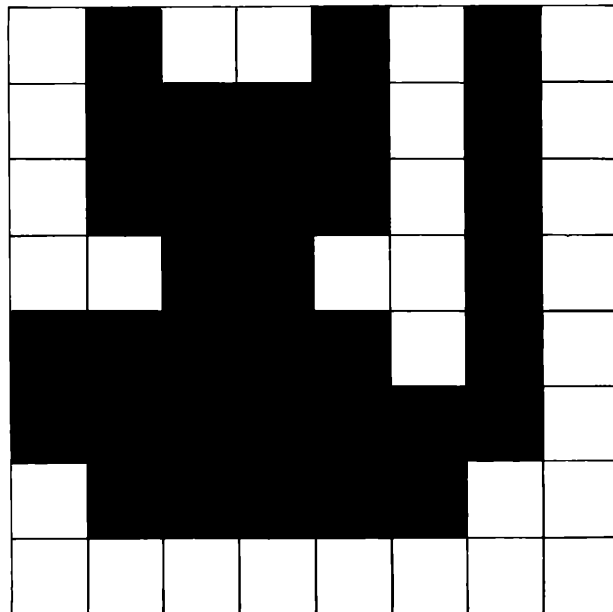



Figura 13.1.

Ora sostituiamo i quadretti neri con degli "1" e gli spazi con degli "0", fino ad ottenere una lista come questa:

```
0 1 0 0 1 0 1 0
0 1 1 1 1 0 1 0
0 1 1 1 1 0 1 0
0 0 1 1 0 0 1 0
1 1 1 1 1 0 1 0
1 1 1 1 1 1 1 0
0 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0
```

Decidete poi quale lettera volete usare: la scelta naturale è "G".

Per fare le cose nel modo più brutale (ma semplice), inserite dalla tastiera i seguenti comandi:

```
POKE USR "G", BIN 01001010
POKE USR "G" + 1, BIN 01111010
POKE USR "G" + 2, BIN 01111010
POKE USR "G" + 3, BIN 00110010
POKE USR "G" + 4, BIN 11111010
POKE USR "G" + 5, BIN 11111110
POKE USR "G" + 6, BIN 01111100
POKE USR "G" + 7, BIN 00000000
```

Se volete, considerate la cosa come una magia! POKE inserisce delle informazioni nella memoria del computer (vedi Capitolo 23 su PEEK e POKE), USR dice solo che si opera su caratteri definiti dall'utente, e BIN sta per "binario". Le cose più importanti sono la "G" (la lettera scelta da voi), i numeri 1, 2,..., 7 che vengono sommati e le sequenze di 0 e 1 copiate, nell'ordine, dalla tabella ottenuta dal disegno del gatto.

Qualsiasi disposizione 8 × 8 di zero e uno può essere trattata in questo stesso modo; e la macchina immagazzinerà fino a 21 differenti caratteri definiti dall'utente per volta. *Non* vengono distrutti da NEW; ma non si possono memorizzare tramite SAVE.

Esistono altri modi di costruire i caratteri. Ho scritto uno dei Programmi preconfezionati (*Costruzione di caratteri*) che vi permetterà di *disegnare* il vostro carattere e quindi di memorizzarlo. Un metodo è quello di convertire i numeri binari in decimale — per esempio tramite comandi diretti del tipo

```
PRINT BIN 01001010
```

che fornisce la risposta 74. Le linee del gatto, in decimale, sono

74, 122, 122, 50, 250, 254, 124, 0

e invece di dare POKE con argomento BIN, potete usare direttamente questi numeri:

POKE USR "G", 74

POKE USR "G" + 1, 122

.....

POKE USR "G" + 7, 0

Ora, scrivendo queste istruzioni in un programma, potete costruire i caratteri durante l'esecuzione; memorizzando il programma con SAVE, in effetti memorizzate anche i caratteri.

Potete anche salvare i caratteri usando la versione di SAVE che agisce su un blocco di memoria e ricaricarli con la relativa LOAD, ma è una cosa un po' troppo avanzata per questo libro.

Naturalmente potete immagazzinare la lista di numeri 74, 122, ecc. sotto forma di matrice; oppure utilizzare il comando DATA (vedi Capitolo 18).

Progetti

1. Costruite i caratteri-utente dei quattro semi delle carte da gioco, memorizzati sotto "C", "F", "Q", "P", usando le griglie di Figura 13.2

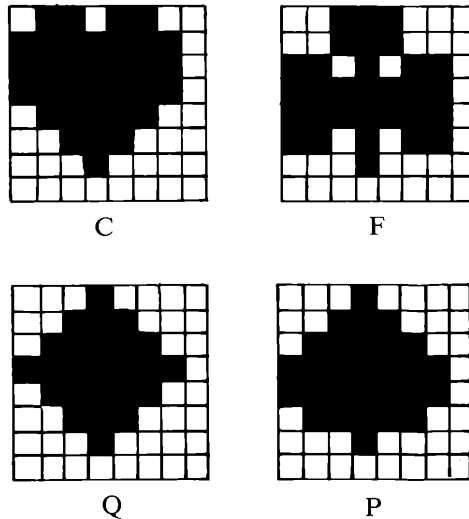



Figura 13.2.

2. Usate la Figura 13.3 per costruire un carattere "1/2".

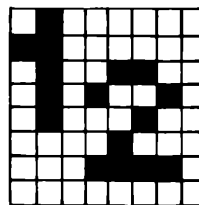



Figura 13.3.

3. Disegnate un set di caratteri per i dodici segni zodiacali.
4. Disegnate due caratteri che rappresentino un cane, differenti solo per la posizione della coda. Stampateli alternativamente in una data posizione, facendo così scodinzolare il cane.
5. Disegnate un uomo che cammina, eventualmente usando diversi caratteri affiancati.

14. Sottoprogrammi

Se potete suddividere un lavoro in più compiti ben definiti, diviene utilizzabile un'arma potente dell'arsenale del programmatore

C'è un aneddoto sui matematici che si adatta ugualmente bene ai programmatori di calcolatori elettronici (daccordo, *sono* animali diversi!). Su come fare bollire dell'acqua. In un primo momento a un matematico viene chiesto di descrivere attraverso quali stadi si debba passare per fare una tazza di té, a partire dal bricco appeso al gancio, in cucina; questi risponde qualcosa del tipo "Togliete il bricco dal gancio, riempitelo d'acqua, posatelo sul fornello, accendete un fiammifero,..." e così via. Gli si domanda poi di descrivere i passi necessari per fare una tazza di té quando il bricco si trovi sul tavolo della cucina, e la risposta diviene "Appendete il bricco sul gancio e procedete come in precedenza".

Quello che fa, è di utilizzare la prima sequenza di operazioni come *sottoprogramma*. La seconda volta, modifica la situazione esistente per rendere applicabile il sottoprogramma, e quindi lo usa come se fosse una singola operazione indivisibile.

Nel gergo dei computer, un sottoprogramma, o subroutine, è uno spezzone di programma che si può scrivere separatamente, e usare ripetutamente come parte di qualche programma più esteso. I sottoprogrammi sono un modo di programmare molto evoluto, dal momento che solitamente rendono più chiara la visione di quello che si sta facendo. È anche più facile il debug di un programma scritto con subroutines, dato che è possibile correggerle singolarmente, e in seguito controllare solo che si colleghino tra loro correttamente.

Il comando di maggior rilievo è GO SUB. È analogo a GO TO, ma molto più versatile. Tipicamente compare in situazioni del genere:

```
100  GO SUB 500
110  varie altre cose
     .....
500  fai qualche cosa
     .....
570  RETURN
```

dove, al solito, le scritte minuscole denotano altre parti del programma. Ecco ciò che accade:

(a) Incontrando la linea 100, il programma salta a 500, *ricordando da dove ha compiuto il salto.*
(b) Esegue poi la 500 e tutto ciò che la segue, fino a incontrare la 570 dove gli viene ordinato RETURN (ritorna).
(c) Quindi torna alla linea originale (nel nostro caso la 100), e continua con l'istruzione *successiva* a quella.

Essenzialmente è tutto come con GO TO, eccetto il fatto che il ritorno al punto di partenza è automatico. La caratteristica di rilievo, tuttavia, è che si può accedere alla stessa subroutine da *diverse* linee dello stesso programma, e anche in questi casi il computer tiene contro di quale fosse la linea di provenienza, linea alla quale ritornerà.

A titolo di esempio darò una spiegazione passo-passo della stesura di un programma che:

(d) Usa i tasti 5, 6, 7, 8 per muovere sullo schermo un cursore [P] .
(e) Stampa al posto del cursore qualsiasi carattere introdotto dalla tastiera.

Per l'introduzione del carattere desiderato dovremo usare INKEY$. Con questo leggeremo il tasto corrente e lo assegneremo alla variabile stringa a$. Vogliamo che il programma reagisca solo ai tasti appena premuti, come si è detto nel Capitolo 12 . Ci serve dunque uno spezzone di programma del tipo:

```
1000  IF INKEY$ <> " " THEN GO TO 1000 ┐
1010  IF INKEY$ = " " THEN GO TO 1010  ├── diventerà la subroutine
1020  LET a$ = INKEY$                  ┘
```

Vengono usati i numeri 1000 e seguenti perché questo diventerà un sottoprogramma e vogliamo posizionarlo ben separato (benché in realtà si risparmino alcune linee di programma ponendo tutte le subroutine all'*inizio* e iniziando l'esecuzione con GO TO invece che RUN — ma si tratta di una raffinatezza su cui non vale la pena soffermarsi in questa sede). Per uscirne nuovamente servirà la linea:

```
1030  RETURN
```

Inoltre, per spostare quel cursore [P] dobbiamo... che cosa? Ebbene, certamente ci servirà sapere *dove* stamparlo; così definiremo due variabili, "a" e "b", che diano il valore della riga e della colonna di stampa. Per evitare che tutto si distrugga prima di cominciare, dovremo assegnare loro dei valori. Il centro dello schermo è un buon posto da cui partire:

```
10  LET a = 10
20  LET b = 15
```

Ora vogliamo usare i tasti 5, 6, 7, 8 per modificare a e b, spostando

così il cursore. La buona vecchia subroutine leggerà la tastiera; così la prossima cosa da fare è ovviamente:

```
30  GO SUB 1000
```

A questo punto a$ ci dirà quale dei tasti 5, 6, 7, 8 sia stato premuto. Vogliamo spostare [P] nella direzione delle quattro frecce su questi tasti. (È questo il motivo per cui usiamo proprio questi tasti: le frecce sono buoni simboli mnemonici!).

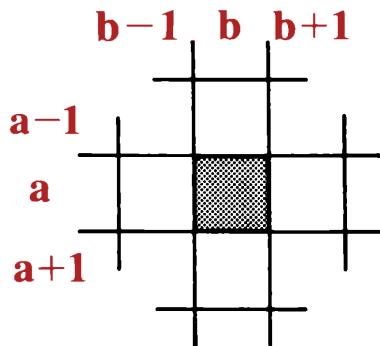



Figura 14.1.

Date uno sguardo alla figura 14.1, che mostra la posizione a, b e le quattro adiacenti. Usando le indicazioni della figura, vediamo che vogliamo che:

il tasto 5 modifichi b in b — 1 e lasci a immutato

il tasto 6 modifichi a in a + 1 e lasci b immutato

il tasto 7 modifichi a in a — 1 e lasci b immutato

il tasto 8 modifichi b in b + 1 e lasci a immutato

Questo è un metodo:

```
40  IF a$ = "5" THEN LET b = b — 1
50  IF a$ = "6" THEN LET a = a + 1
60  IF a$ = "7" THEN LET a = a — 1
70  IF a$ = "8" THEN LET b = b + 1
```

Avendo mosso il cursore, ci piacerebbe vedere dove è andato; e, per divertimento, lo faremo lampeggiare:

```
80  PRINT AT a, b; FLASH 1; "P"
```

E adesso? Vogliamo inserire un carattere da stampare al posto di questo [P]. Ancora la subroutine!

```
90  GO SUB 1000
```

Questa volta la macchina legge la tastiera, assegna ad a$ il valore

che viene trovato (qualsiasi tasto premiamo, a, b, c, d,... 7, 3, >,...) e ritorna *alla prima linea successiva alla linea 90*. Vogliamo che questa dica di stampare il carattere che è stato trovato:

```
100  PRINT AT a, b; a$
```

Quasi finito. Per ora, tutto questo funziona solo una volta. Vogliamo tornare all'inizio e ricominciare, mantenendo però i nuovi valori di a e b e *non* riportandoli a 10 e 15. Facile:

```
110  GO TO 30
```

(30 rispedisce immediatamente a 1000. Perché non avrebbe funzionato 110 GO SUB 1000?).

Scrivete tutte le linee precedenti (ci siamo assicurati che i numeri compaiano nel giusto ordine, cosa che non sempre si verifica al primo tentativo quando scrivete un programma, quindi attenzione!) e premete RUN. Non succederà niente; ma se premete uno qualsiasi dei tasti 5, 6, 7, 8, vedrete apparire il cursore [P] nella sua nuova posizione. (Premendo qualche altro tasto, appare alla vecchia posizione a, b — una caratteristica positiva che non avevamo effettivamente programmato). Premete quindi un tasto, per esempio t. [P] sparisce, per essere sostituito da una t. Spostate il cursore (ora invisibile) usando i tasti 5, 6, 7, 8; stampate il prossimo carattere; e così via. Potete scrivere su tutto lo schermo. (Che ve ne pare di un programma per la costruzione di parole crociate al computer?).

Per evitare di schiantarsi sui lati dello schermo, un minimo di protezione è una buona idea:

```
75  IF a < 0 OR a > 21 OR b < 0 OR b > 31 THEN GO TO 30
```

e senza dubbio riuscirete a escogitare altri perfezionamenti.

Progetti

Invece della subroutine con INKEY$, generate sia gli ingressi 5, 6, 7, 8 che il carattere da stampare in modo casuale; fate funzionare il tutto e attendete gli sviluppi. Per stampare caratteri casuali usate PRINT CHR$ INT (65 + 26 * RND), che sceglie a caso un carattere dall'alfabeto. (Perché?).

Ecco un altro esempio — un'introduzione alla Computer Art. Disegna quadrati a caso, neri o a scacchi, finché non esaurisce la memoria o voi non lo fermate con BREAK.

```
10  LET a = 10 * RND
20  LET b = 10 * RND
30  LET q = 5 * RND
40  LET r = 5 * RND
50  LET k = INT (2 * RND)
60  IF k = 0 THEN LET m$ = "■"
```

```
  70  IF k = 1 THEN LET m$ = "▛"
  80  GO SUB 1000
  90  GO TO 10
1000  FOR i = a TO a + q
1010  FOR j = b TO b + r
1020  PRINT AT i, j; m$
1030  NEXT j
1040  NEXT i
1050  RETURN
```

Progetto: dadi grafici

Generate un numero casuale compreso tra 1 e 6 servendovi di INT (1 + 6 * RND). A seconda del numero generato, diciamo n, stampate al centro dello schermo n punti disposti proprio come lo sono sulla faccia di un dado. Ripetete l'operazione a ogni pressione di ENTER.

Allo scopo, studiate un sottoprogramma per ciascuna delle sei possibilità. Se la subroutine per n è scritta in modo da iniziare alla linea, per esempio, 500 + 100 * n (cioè a 600, 700, ecc.) e finire sicuramente prima dell'inizio di quella successiva, potete usare GO SUB 500 + 100 * n per evitare complessi salti condizionali. Avrete comunque bisogno di sei comandi RETURN diversi.

Per la ripetizione tramite ENTER, usate una linea che richieda l'ingresso di un carattere: INPUT k$; fatela seguire da un comando GO TO che rispedisca tutto verso l'inizio. k$ non ha nessun utilizzo, ma la macchina lo attende, prende ENTER, continua fino al GO TO. Chiaro?

15. Son et lumière

Lo Spectrum canta con il Blues

Provate a far girare questo programmino:

```
10  FOR i = 1 TO 6
20  INK i
30  CIRCLE 100, 80, i * 10
40  BEEP 0.5, i
50  NEXT i
```

Non è terribilmente eccitante, forse, ma illustra un paio di cose. In primo luogo possiamo vedere come si modifica il colore del simbolo da disegnare (o stampare). Dobbiamo solo specificare il numero associato al colore desiderato in un'istruzione INK (''inchiostro''). Non è necessario ricordare la corrispondenza tra numeri e colori, dato che questi ultimi compaiono sulla tastiera sopra le relative cifre. Perciò nel programma, la prima volta che si esegue la linea 20, si ha i = 1 e il comando INK 1 viene interpretato dal BASIC come ''usare inchiostro blu per disegni e stampe, fino a nuovo ordine''. Naturalmente l'unica cosa tracciata mentre l'''inchiostro'' è blu è il cerchio più piccolo. Nel momento in cui viene disegnato il secondo cerchio, la linea 20 è diventata equivalente a INK 2 e perciò si usa il rosso, e così via.

Dopo il completamento di ogni cerchio, il computer emette un bip di trionfo. È l'effetto della linea 40. Il primo valore dopo BEEP dà la durata della nota in secondi (così in questo programma ciascuno di questi suoni dura 0.5 secondi — sembrava più lungo, no?) e il secondo identifica la nota da suonare. Se questo valore è zero, la nota è il do centrale. 1 è do#, 2 è il re, 3 re#, 4 mi, 5 fa (non esiste il mi#!) e così via. Valori negativi vi portano sotto il do centrale: —1 è si, —2 è la# ecc.

L'impiego più semplice di BEEP è per segnalare all'utente che è successo qualcosa (come un errore nel programma) o che il computer sta aspettando, o ha accettato, un ingresso. Naturalmente si può

usare per suonare della musica, ma voglio soprassedere all'esame di questa possibilità, per un poco (vedi Programmi preconfezionati), e trattare qui alcune altre caratteristiche del colore.

In primo luogo non è solo il colore dell'"inchiostro" che è possibile cambiare. Lo sfondo (che il BASIC — piuttosto sensatamente — chiama PAPER, cioè "carta") può essere uno qualsiasi degli otto colori. Naturalmente se INK e PAPER sono del medesimo colore, non vedrete assolutamente niente, cosa che può generare un poco di confusione se, per esempio, premete LIST dopo un programma che ha modificato il colore di INK rendendolo uguale a quello di PAPER. (Menziono questo caso perché mi succede spesso e io ogni volta credo, erroneamente, che il programma sia stato cancellato).

Ora potreste aspettarvi di essere in grado di scrivere PAPER 2 e avere uno sfondo rosso, ma se inserite il comando nel programma, diciamo alla linea 5, durante l'esecuzione non ha luogo alcun cambiamento. Il motivo è che il sistema non è in grado di cambiare il colore dello sfondo vi appare se una qualche scritta, e non ha modo di essere certo che lo schermo sia vuoto se non ha appena eseguito una CLS (cancellazione schermo). Così la macchina risponderà all'istruzione PAPER solo dopo aver incontrato una CLS.

È possibile inoltre cambiare il colore del contorno (per esempio scrivendo BORDER 4 per ottenere un contorno verde) e, in questo caso, il cambiamento è immediato. Inserendo:

```
25  BORDER i
```

il contorno sarà dello stesso colore del cerchio che viene disegnato.

Modificate la linea 25 in modo che sia:

```
25  BORDER 7 — i
```

Questo è un poco più di dubbio (per non dire di cattivo) gusto, vero?

Aggiungete ora le linee:

```
 60  INK 4
 70  PLOT 100, 0
 80  DRAW 0, 175
 90  PLOT 0, 80
100  DRAW 255, 0
```

e fate eseguire ciò che ne risulta.

Vengono tracciate, come vi aspettereste, due linee incrociate verdi, ma se guardate più da vicino, vedrete che ora nella zona dell'intersezione con le linee le circonferenze sono state colorate di verde. Non è un guasto del vostro televisore, né un baco dello Spectrum. È una caratteristica di come lo Spectrum gestisce lo schermo (nel gergo dei computer, un baco che non si può uccidere viene sempre chiamato "caratteristica". Così sembra meno indesiderabile). In ogni modo, la ragione di questo strano comportamento è che gli attributi di un punto dello schermo (cioè il suo colore, la luminosità e il fatto

di lampeggiare o meno) non sono limitati a un solo pixel, ma fanno riferimento a un intero carattere; che, come abbiamo già visto, combina 64 pixel, disposti in un quadrato di 8 × 8. Di conseguenza, quando cambiate il colore di un pixel, anche tutti gli altri nella stessa regione di 8 × 8 seguono la stessa sorte.

Nella maggior parte dei casi, tutto questo porterà ad ottenere dei risultati comunque accettabili. Cambiare la luminosità del quadrato di un carattere generalmente non presenta alcun problema, e la cosa si ottiene usando il comando BRIGHT 1, per aumentare la luminosità, e BRIGHT 0 per ridurla nuovamente. Una volta che è stato eseguito un comando BRIGHT 1, tutto viene stampato ad alta luminosità, finché non si raggiunge il successivo BRIGHT 0. Se dunque volessimo che i due cerchi più interni siano stampati più luminosi dei rimanenti, potremmo aggiungere i comandi:

```
 5  BRIGHT 1
23  IF i > 2 THEN BRIGHT 0
```

Il lampeggiamento è un poco diverso. Potete attivare e disattivare il lampeggiamento con FLASH 1 e FLASH 0, proprio come si usa BRIGHT. Tuttavia, a causa del problema degli attributi del quale si è appena detto, se provate con il programma per il tracciamento di cerchi, cominciano a lampeggiare grossi blocchi dello schermo in modo snervante. Provate

```
4  FLASH 1
6  CLS
```

per capire quello che intendo dire. (Come PAPER, anche FLASH richiede CLS per l'attivazione). Potreste anche escogitare un uso per questo tipo di figura, ma l'unico che riesco a immaginare è l'induzione di emicrania nelle cavie da laboratorio.

In genere, quindi, FLASH si usa soprattutto nelle istruzioni PRINT, dove vogliamo far lampeggiare l'intero quadrato del carattere, piuttosto che nelle PLOT, DRAW e CIRCLE dove gli effetti non sono probabilmente altrettanto gradevoli.

Ancora meglio, tutti i vari INK, PAPER, FLASH eccetera, si possono incorporare in un'istruzione PRINT, nel qual caso il loro effetto è limitato ai simboli visualizzati da quel singolo comando, e non vi serve una CLS per attivarli. Per esempio, potete scrivere

```
110  PRINT AT 10, 0; INK 5; FLASH 1; PAPER 4; "xxx"
```

e quindi comandare RUN (dopo aver cancellato la linea 4 ed eseguito FLASH 0: CLS sotto forma di comando diretto per sbarazzarvi del lampeggiamento): lampeggeranno solo le x. Se ora scrivete LIST vedrete che l'inchiostro è ancora verde(linea 60) benché a 110 fosse stato posto a ciano; inoltre il listato non lampeggia.

Non ho trattato tutti i modi di operare con il display possibili con lo Spectrum, e non ho intenzione di farlo. Questo è un testo introduttivo e la questione è di non opprimervi con una enorme quantità

di dettagli. Quando sarete paghi di queste tecniche, vorrete ovviamente avventarvi sul resto del *Manuale*, e a quel punto non dovreste avere molte difficoltà.

Tre questioni da puntualizzare per finire. Primo, la necessità di pulire lo schermo prima che degli attributi abbiano effetto non significa in realtà che dovete inserire necessariamente CLS. Per esempio, premere il tasto ENTER senza che niente lo preceda avrà lo stesso effetto di CLS: LIST. Così, invece di premere FLASH 0: CLS per eliminare il lampeggiamento, potete scrivere FLASH 0 e poi premere ENTER due volte invece che una come al solito.

Secondo, avrete notato che la regione della cornice subito sotto lo sfondo si comporta stranamente: non sempre cambia come dovrebbe oppure non conserva il proprio colore all'inizio di un'esecuzione. Potreste avere già compreso che questa zona corrisponde alla linea dei comandi e dei messaggi (cioè dove il BASIC comunica con voi). È questo il problema: il BASIC non vuole cadere nella trappola che ho menzionato in precedenza di scrivere con inchiostro verde su carta verde. È, dopo tutto, piuttosto importante che sappiate quando sta cercando di dirvi qualcosa. Infatti modifica il suo colore tra bianco e nero a seconda del colore del bordo, in modo che il risultato sia il più chiaro possibile. Dunque questa zona ha un colore fisso durante ogni singola esecuzione. Se trasformate l'intero programma in un ciclo inserendo

```
120  RUN
```

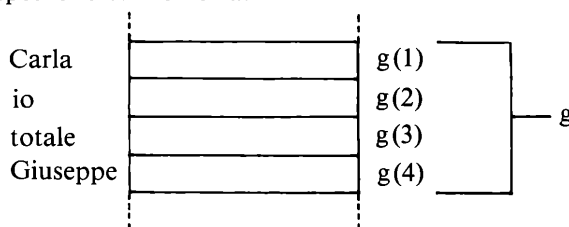
e comandando poi dei BREAK e CONT in diversi punti, vedrete questo effetto in modo molto chiaro.

In terzo luogo, una questione di stile. Quando scrivete FLASH 1, state attivando l'attributo di lampeggiamento. Quanto più comodo sarebbe se poteste scrivere "FLASH sì" e "FLASH no", oppure "BRIGHT sì" e "BRIGHT no" invece di usare gli 0 e 1 che sono piuttosto privi di significato. Presto fatto. Inserite una linea 1:

```
1  LET sì = 1 : LET no = 0
```

Ora, quando scrivete "FLASH sì", il BASIC sostituirà l'uno a "sì" (o lo zero a "no") e il programma è più leggibile

Matrici

È possibile che un gruppo di celle di memoria siano poste in relazione tra loro dal fatto di avere lo stesso nome. Un sì fatto gruppo è detto matrice o *array*. Possiamo considerarlo nel modo seguente. Ecco uno spezzone di memoria:

Solitamente daremmo a ogni cella un nome BASIC (Carla, io, ecc.) come indicato sulla sinistra. Possiamo invece chiamare tutte le celle con lo stesso nome (ho scelto "g") come mostrato a destra. Se adottiamo questo approccio (e per il momento non sarà chiaro il perché dovremmo), ci sono tre cose da tenere bene in mente:

1. Il nome di una matrice è una singola lettera, così non si possono avere matrici chiamate t2 o Alberto.
2. Le matrici possono essere lunghe a piacere (entro i limiti di memoria della macchina) così, prima di usarle, è necessario dire al BASIC quanto ognuna sia grande. Esiste allo scopo una istruzione DIM (abbreviazione per "dimensione") che avrà questo aspetto per la matrice del disegno:

   ```
   10  DIM g (4)
   ```

   In altri termini, è definito il nome della matrice, insieme al numero di celle che contiene posto tra parentesi.
3. Sarà necessario poter identificare una data cella all'interno dell'array, e la tecnica relativa è indicata nella figura: ci si riferisce alla prima cella con g (1), alla seconda con g (2) e così via.

Vediamo per prima cosa come si possa inserire in una matrice un insieme di 20 numeri. Il modo più semplice sarebbe scrivere:

```
 10  DIM n (20)
 20  INPUT n (1)
 30  INPUT n (2)
 40  INPUT n (3)
     .....
210  INPUT n (20)
```

ma è ovviamente tedioso e non c'è chiaramente alcun vantaggio rispetto all'uso di diversi nomi di variabile, dal momento che ogni elemento della matrice viene singolarmente nominato nelle istruzioni da 20 a 210.

Il trucco, tuttavia, consiste nel fatto che il contenuto delle parentesi non deve necessariamente essere un numero. Può essere il nome di una variabile. Possiamo così parlare di n (p), per esempio, e significherà n (2) se p = 2, n (17) se p = 17.

Ora il problema è semplice. Si può osservare che il valore tra parentesi va da 1 a 20 in passi unitari, ed è un indizio per un ciclo FOR:

```
10  DIM n (20)
20  FOR p = 1 TO 20
30  INPUT n (p)
40  NEXT (p)
```

La prima volta lungo il ciclo si ha p = 1 e la linea 30 è equivalente a INPUT n (1). La seconda volta p = 2 e la linea 30 diviene INPUT n (2), e così via. Ho deliberatamente scelto il nome "p" della variabile, dato che "punta" all'elemento della matrice a cui si riferisce per tutto il tempo, e questo è un modo molto utile di considerare la manipolazione di una matrice.

CHR$ musicale[1]

Facciamo ora qualcosa di interessante con le matrici. Abbiamo già incontrato BEEP e sottolineato come sia possibile eseguire della musica per suo tramite, benché al livello più elementare risulti un compito piuttosto noioso, dal momento che dovremmo calcolare i numeri corrispondenti a ogni nota per poi inserire tutta una serie di comandi BEEP con il numero appropriato.

Perché non fare che il computer si guadagni da vivere eseguendo la traduzione da note a numeri in nostra vece?

Per cominciare manteniamo le cose a un livello semplice e lavoriamo solo entro un'ottava, quella intorno al do centrale. I codici numerici e la relativa notazione anglosassone sono:

A = la	: —3		D# = re#	:	3
A# = la#	: —2		E = mi	:	4
B = si	: —1		F = fa	:	5
C = do	: 0		F# = fa#	:	6
C# = do#	: 1		G = sol	:	7
D = re	: 2		G# = sol#	:	8

Dal momento che i diesis hanno sempre un valore superiore di uno a quello della corrispondente nota naturale, non è necessario tabularli, così le informazioni fondamentali che ci servono sono

A = la	: —3		E = mi	:	4
B = si	: —1		F = fa	:	5
C = do	: 0		G = sol	:	7
D = re	: 2				

---

[1] NdT. Lo scopo didattico di questo paragrafo è di introduzione all'uso delle variabili indirizzate e l'ordinata notazione musicale anglosassone ben si prestava allo scopo, per essere ogni nota naturale individuata da un singolo carattere e per essere questi ultimi in ordine alfabetico, cosa che conduce a dei codici interni adiacenti e crescenti. Per questi motivi, e per il fatto che l'uso dei nomi latini delle note avrebbe comportato (1) l'introduzione di almeno due lettere per ogni nota, più il diesis e (2) l'uso di espressioni del tipo IF n$ = "la" THEN LET p = 1 per il calcolo del puntatore nella matrice (e a questo punto, perché non fare IF n$ = "la" THEN LET nota = —3, seguito in esecuzione da BEEP 0.5, nota?), vanificando così l'effettiva utilità dell'organizzazione dei dati in forma matriciale, è stata mantenuta la notazione anglosassone. Tuttavia, oltre agli appunti relativi al programma *Compositore*, a cui vi rimandiamo, si suggerisce di usare una matrice contenente i nomi italiani delle note, per farli stampare ordinatamente durante l'esecuzione, magari in colori diversi. E che dire di una subroutine che faccia comparire su un pentagramma la partitura in esecuzione, disegnando le note tramite dei caratteri-utente da definire?

Non c'è alcun nesso tra questi numeri, ha dunque senso conservarli in una matrice per leggere il loro valore quando è necessario.

Memorizzarli è facile; si potrebbe scrivere:

```
10  DIM s (7)
20  LET s (1) = —3 : LET s (2) = —1 : LET s (3) = 0 : LET s (4) = 2
30  LET s (5) = 4 : LET s (6) = 5 : LET s (7) = 7
```

(Esistono altri metodi più soddisfacenti se la matrice fosse più grande; ma questo è adeguato al nostro caso).

Ora inseriamo una nota (tramite la lettera del suo nome anglosassone) dalla tastiera, ma a questo punto abbiamo bisogno di un qualche sistema per "andare a vedere" il corrispondente valore numerico nella matrice s. In altre parole vogliamo qualche cosa del tipo

```
 40  INPUT "Inserite una nota"; n$
     ..... [Qualche meccanismo che generi il valore di p
           partendo da n$ (p. es. se n$ = "A", p = 1;
           se n$ = "b", p = 2; ecc.)]
100  BEEP 0.5, s (p)
```

Potremmo naturalmente scrivere una serie di comandi come:

```
45  IF n$ = "A" THEN LET p = 1
50  IF n$ = "B" THEN LET p = 2
```

e la cosa funzionerebbe, ma è disordinata. Un modo migliore è fare uso dei codici numerici interni dei caratteri. "A" è in realtà memorizzato sotto forma del numero 65, "B" come 66 e così via. (Vedi Appendice A del *Manuale*). Potete scrivere

```
LET p = CODE "A"
```

per accedere al codice di A. Così ponendo

```
50  LET p = CODE n$
```

otterremmo i valori 65, 66 ecc. Con i nomi anglosassoni delle note, A, B, C... ecc., si ottengono dei valori di p maggiori di 64 rispetto a quelli ricercati. Così la linea 50 potrebbe essere:

```
50  LET p = CODE n$ — 64
```

Aggiungiamo una linea per tornare all'istruzione di ingresso

```
110  GO TO 40
```

e abbiamo ottenuto una tastiera musicale estremamente primitiva.

(Non dimenticate le maiuscole per la notazione anglosassone).

La prima cosa sbagliata è che abbiamo ignorato i diesis. Così se scrivete "A#" otterrete ancora "A"; dato che la funzione CODE opera sulla prima lettera di una stringa, non noterà nemmeno la presenza di '#'!

È necessario esaminare il secondo simbolo della stringa, n$ (2):

```
60  IF n$ (2) = '#' THEN LET i = 1
```

Possiamo poi modificare la linea 100 per avere

```
100  BEEP 0.5, s (p) + i
```

in modo che la nota da suonare sia incrementata di 1 se i = 1. Naturalmente, i deve essere posto a 0 dall'inizio del ciclo, altrimenti, una volta incontrato il primo diesis, i rimarrà a 1 e tutte le note verranno trattate come fossero diesis:

```
45  LET i = 0
```

Sfortunatamente le cose sono complicate dal fatto che n$ (2) non esiste se introducete una nota naturale (solo B, per esempio). Così come stanno le cose il programma funziona se premete C#, A#, G#, ma ottenete un messaggio di errore non appena provate con C.

La facile via d'uscita da questa difficoltà è quella di definire n$ come matrice di lunghezza 2:

```
11  DIM n$ (2)
```

Ora n$ (2) esiste sempre, e anche se non viene usato il secondo carattere, il BASIC completa la stringa con uno spazio vuoto.

La seconda cosa che non funziona con il nostro strumento musicale è che suona le diverse note in modo incostante a seconda della vostra abilità con i tasti e delle caratteristiche della tastiera (dopo tutto lo Spectrum non è concepito come organo elettronico). Invece di suonare una nota non appena inserita, perché non immagazzinarla (avete indovinato - in una matrice) e suonare l'intero brano quando tutte le sue note sono in macchina? Avremmo anche il vantaggio di poter ripetere la melodia a piacimento.

Per prima cosa ci serve una matrice dove tenere le note codificate:

```
12  DIM t (1500)
```

Il nostro motivo può avere fino a 1500 note.

Il ciclo d'ingresso cambia un poco, dato che il numero di note non è piú illimitato, e anche perché il valore introdotto viene memorizzato, non suonato. Inoltre ci servirà una via d'uscita dal ciclo, se ci sono meno di 1500 note.

Le linee 20 e 30 non si modificano, e naturalmente rimangono fuori dal ciclo, dunque iniziamo dalla linea 35:

```
35  FOR q = 1 TO 1500
```

Dopo INPUT alla linea 40, servirà un test per vedere se la sequenza è già terminata. Usiamo "* *" per finire, scrivendo la linea

```
42  IF n$ = "* *" THEN GO TO 200
```

per portarci fuori dal ciclo.

La linea 100 deve modificarsi per immagazzinare la nota codificata invece di suonarla:

```
100  LET t (q) = s (p) + i
```

Dal momento che q assume i successivi valori 1, 2, ..., la prima nota verrà memorizzata in t (1), la successiva in t (2) e così via.

Il ciclo termina a 110. Sostituiamo GO TO con:

```
110  NEXT q
```

Ora possiamo iniziare a suonare il motivo dalla linea 200 in avanti:

```
200  FOR r = 1 to q
210  BEEP 0.5, t (r)
220  NEXT r
```

[Nota: questo ciclo va solo fino a q, non più 1500. Questo perché quando si lascia il ciclo precedente, q contiene il numero di note del motivo.]

Per continuare a ripetere, tutto quello che dobbiamo aggiungere è:

```
230  GO TO 200
```

Nel caso le continue modifiche e revisioni per cui siamo passati vi abbiano confuso, troverete il listato completo del programma finale tra i Programmi preconfezionati.

Alla fine, aggiungiamo uno schermo colorato che accompagni la musica. Forse la cosa più semplice che possiamo fare è modificare il colore della cornice in accordo con la nota che viene suonata. Ora, sarebbe bello scrivere

```
215  BORDER t (r)
```

così che ogni nota corrisponda a un colore della cornice, ma non è possibile, perché l'estensione dei valori che si possono trovare in t (r) è da —3 a 8, mentre il campo dei colori possibili va solo da 0 a 7. Sommando 3 a t (r) l'estensione diventa 0-11, se poi il risultato viene moltiplicato per 7/11, otteniamo dei valori tra 0 e 7 come desiderato. Così:

```
215  BORDER INT ((t (r) + 3) * 7/11)
```

Cambiamo il colore dello sfondo, ogni volta che viene suonata una nota, in questo modo:

```
216  PAPER 8 * (r/8 — INT (r/8)): CLS
```

Questo rallenta un poco le cose, non è vero?
Sostituite 216 con:

```
216  PRINT INK ((t (r) + 3) * 7/11); "□□□";
```

(Non è assolutamente necessario imporre INT all'espressione; il BASIC lo farà in ogni caso, perché INK non può avere associato un valore non intero).
Sperimentate; probabilmente potete escogitare un display ancora più orribile!

16. Debugging III

...ma perché non usare lo Spectrum per correggere se stesso?

Nell'ultimo esempio, il processo di correzione è stato ottenuto tramite una considerevole quantità di gravoso lavoro da parte nostra. Sarebbe bello fare sì che la macchina, la cui sola funzione fino ad ora è stata di rimanere lì a darci la risposta sbagliata con aria di sufficienza, facesse in nostra vece una parte del lavoro.

La domanda è: che genere di cose può utilmente dirci la macchina sul modo in cui sta eseguendo il programma? Sono da considerare tre aspetti principali:

1. Che valori ottiene il programma per le diverse variabili, nei vari stadi dell'esecuzione?
2. Dove va il programma? Cioè, quali linee vengono eseguite e in quale ordine?
3. Quanto spesso ci va? Cioè, quante volte vengono eseguite particolari linee o gruppi di linee?

Alcune macchine offrono automatismi per la stampa di almeno qualcuna di queste informazioni, ma non si può (non ancora?) avere tutto per poche centinaia di migliaia di lire, così dobbiamo inserire qualche istruzione extra nei nostri programmi, per provvedere a questi dettagli. Diamo uno sguardo ai paragrafi che seguono.

Stampa del valore delle variabili

È molto facile stampare un valore di variabile dovunque vogliamo. Tutto quello che dobbiamo fare è inserire un'istruzione PRINT in un'opportuna posizione del programma. Per esempio, nel nostro programma della media, potremmo introdurre una linea

```
55   PRINT c
```

che ci permetterebbe di seguire i cambiamenti del valore di c durante l'esecuzione. (In effetti non sarebbe difficile fare sì che la macchina fornisca una copia della tabella di esecuzione che abbiamo prodotto a mano — potreste provarci).

In questo caso il vero problema è la scelta di dove porre in uscita i valori intermedi in modo sensato ed efficiente, altrimenti otterrete

solo enormi quantità di numeri che richiedono tanto tempo per l'analisi quanto un'esecuzione manuale.

Nel caso della seconda esecuzione del problema della media, ricorderete che era stato prodotto un messaggio d'errore alla linea 100, senza che fosse stampato alcun valore. Un primo passo molto sensato sarebbe d'includere una linea 95:

```
95  PRINT s, c
```

Non dimenticate che la linea 40 richiederà una modifica:

```
40  IF n < 0 THEN GO TO 95
```

Altrimenti la nuova linea non sarà mai eseguita. Questo avrà l'effetto di confermare il vostro sospetto che c contiene zero, ma niente di più. Aggiungete ora la linea 65

```
65  PRINT c, i, n, s, z
```

così da ottenere una lista completa di tutte le variabili (per praticità in ordine alfabetico) alla fine di ogni ciclo. Si creerà una versione semplificata della tabella di esecuzione, che indicherà ancora i punti salienti.

Tra parentesi, ecco un facile trucco: nel procedimento di prova del programma, potreste voler togliere temporaneamente una delle nuove linee, per evitare di avere stampati troppi valori in una volta. Questo significherebbe riscrivere più tardi l'intera linea o, peggio, come nel caso della nostra linea 95, modificare anche un'altra linea, dato che a 40 dovremmo porre ancora

```
40  IF n < 0 THEN GO TO 100
```

se venisse tolta la 95. Questo non è necessario. Scrivete semplicemente REM all'inizio di ogni linea che volete inattivare. In questo modo la linea 95 diventa:

```
95  REM PRINT s, c
```

Dato che ora si tratta di un commento la macchina lo ignorerà, ma rimane ben legale il salto a 95! Quando volessimo ancora quell'istruzione, tutto quello che faremmo sarebbe cancellare REM.

Ripercorrere la strada

Il modo più semplice, per rintracciare la strada seguita da un programma, è quello di far seguire ogni linea da un comando PRINT che semplicemente stampi il numero della linea appena eseguita. Per esempio, il programma della media diventerebbe:

```
10  LET s = 0
11  PRINT "10"
```

```
20  LET c = 0
21  PRINT "20"
30  INPUT n
31  PRINT "30"
    ecc.
```

Ancora una volta corriamo il rischio di produrre troppe informazioni e di non essere capaci di vedere il nocciolo della questione, così cerchiamo di essere più selettivi riguardo questo procedimento di "inseguimento". Il tipo di domande a cui risponderà la procedura è "Il programma prende una nuova strada quando ci aspettiamo che lo faccia?". In questo caso ha senso restringere il nostro inseguimento alle sole zone dove si trovano le decisioni.

Per esempio, supponiamo che un sottoprogramma abbia il compito di introdurre un giorno di un mese. Questo valore dev'essere nel campo 1-31, così sarebbe una buona pratica programmativa assicurarsi che l'utente abbia inserito un valore lecito, prima di continuare. Potremmo scrivere qualcosa del genere:

```
 50  INPUT g
 60  IF g > 0 OR g < 32 THEN GO TO 200
 70  REM qui per un giorno non valido
     .....
200  REM qui per un giorno valido
     .....
```

Il programma non si comporta come dovrebbe, così aggiungiamo delle istruzioni per lasciare delle tracce dopo le linee 50, 70 e 200:

```
 50  INPUT g
 51  PRINT g "*50*"
 60  IF g > 0 OR g < 32 THEN GO TO 200
 70  REM qui per un giorno non valido
 71  PRINT "*70*"
     .....
200  REM qui per un giorno valido
201  PRINT "*200*"
     .....
```

Troviamo che per qualsiasi ingresso, la traccia risultante è comunque sempre:

```
*50* *200*
```

(Sto usando gli asterischi in modo che i numeri che segnano la trac-

cia non possano essere confusi con i numeri stampati del programma. Qualsiasi carattere speciale che attiri la vostra attenzione farà al nostro caso).

Dunque non si può fare arrivare il programma a linea 70. C'è solo una conclusione sensata: la condizione g > 0 OR g < 32 è sempre soddisfatta. Logico che sia così — qualsiasi numero è o maggiore di zero o minore di 32. Quello che avremmo dovuto dire era:

```
60  IF g > 0 AND g < 32 THEN GO TO 200
```

Confondere AND e OR è un errore comune, a causa del modo piuttosto disordinato in cui usiamo queste parole nel linguaggio comune. In questo caso bisogna soddisfare entrambe le condizioni per avere un giorno valido e ciò richiede la clausola AND.

Profili di programmi

Un profilo di programma mostra quante volte sia stata eseguita ogni linea di un programma. Come al solito, l'informazione è ridondante, e dobbiamo scegliere le parti di programma che vogliamo sezionare. È cosa abbastanza facile da fare. Supponete che vogliamo scoprire quante volte viene eseguita la linea 420 di un particolare programma. Predisponiamo a zero un contatore all'inizio del programma e quindi lo incrementiamo di 1 ogni volta che si attraversa la linea 420.

```
  5  LET cp = 0
     .....
420  LET a = a * (p — 1)
421  LET cp = cp + 1
     .....
809  PRINT cp
810  STOP
```

Vediamo un esempio concreto. Il programma che segue è inteso ad accettare fino a un massimo di 20 numeri, con zero come terminatore, e a ordinarli per valori crescenti.

Se la sequenza d'ingresso è:	l'uscita risultante dovrebbe essere:
3	1
8	2
1	3
4	4
2	8
0	

Lo zero non dovrebbe apparire dal momento che è solo un terminatore.

```
 10  DIM a (20)
 20  FOR p = 1 TO 20
 30  INPUT a (p)
 40  IF a (p) = 0 THEN GO TO 60
 50  NEXT p
 60  LET n = p
 65  LET f = 0
 70  FOR p = 1 TO n
 80  IF a (p) < a (p + 1) THEN GO TO 130
 90  LET t = a (p)
100  LET a (p) = a (p + 1)
110  LET a (p +1) = t
120  LET f = 1
130  NEXT p
140  IF f = 1 THEN GO TO 65
150  FOR p = 1 TO n
160  PRINT a (p)
170  NEXT p
```

Il programma non esegue il suo compito (battetelo e provate). Infatti cade in un ciclo infinito.

Da dove cominciare a guardare? Il primo ciclo (20-50) sembra abbastanza innocuo, mentre quello finale (150-170) serve solo a stampare i contenuti della matrice, a. Sembra sensato concentrarsi sul ciclo che va da 70 a 130. Dalla linea 80 è chiaro che alle volte sono eseguiti tutti i comandi del ciclo, mentre altre volte quelli da 90 a 130 vengono ignorati. Inseriamo allora due contatori, c1 e c2, che daranno un profilo del programma contando rispettivamente il numero di volte che si entra nel ciclo e il numero di volte che se ne esegue l'ultima parte.

Possiamo arrivare a ciò con:

```
 67  LET c1 = 0
 68  LET c2 = 0
 75  LET c1 = c1 + 1
125  LET c2 = c2 + 1
132  PRINT c1, c2
```

Visto che ci siamo, potremmo anche stampare i contenuti della matrice alla fine di ciascun ciclo, dal momento che è ovvio che al suo

interno i numeri vengono rimescolati e che in totale sono in gioco solo poche altre variabili.

```
134   FOR q = 1 TO n       [Dato che quello da 1 a n sembra
135   PRINT a (q);          essere lo spezzone rilevante
136   NEXT q                della matrice]
137   PRINT
```

Proviamo alcuni insiemi di dati in ingresso per vedere che cosa succede: se introduciamo

```
3
6
1
8
5
0
```

otteniamo

```
6  4
316500
6  4
130056
6  2
100356
6  2
001356
6  2
001356
6  1
001356
6  1
001356
```

e così via, fino all'esaurimento della memoria.

Ebbene, sembra porre in ordine i valori, ma abbiamo perso l'"8" e da dove viene quella coppia di zeri? Inoltre, percorre costantemente il ciclo principale per 6 volte, ma il numero di iterazioni per il ciclo ridotto continua a diminuire fino a 1, e lì rimane.

Uno degli zeri è ovviamente il terminatore, e l'altro è un elemento della matrice che non è stato assegnato durante l'esecuzione, ma è stato inizializzato a zero dal sistema. In altre parole, il programma

sta trattando due valori di troppo. Così riscriviamo la linea 60

```
60  LET n = p — 2
```

e riproviamo. La speranza è l'ultima a morire...
Otteniamo:

```
4      2
3165
4      2
1356
4      0
1356
1
3
5
6
```

C'è un certo progresso: abbiamo eliminato quegli zeri. Ma non ci ha ancora restituito il nostro 8.

È difficile capire come possa essere andato perso. Forse è ancora lì, ma non stampato. Rivediamo le linee 150-170. Dev'essere troppo ristretto il campo 1-n. Aumentiamolo di 1:

```
150  FOR p = 1 TO n + 1
```

E, naturalmente, visto che ci siamo, la linea di prova 134 ha presumibilmente lo stesso problema. Mettiamo a posto anche quella:

```
134  FOR q = 1 TO n + 1
```

Ancora una volta all'assalto, cari amici, ancora una volta...
Ora abbiamo (per gli stessi dati):

```
4      2
31658
4      2
13568
4      0
13568
1
3
5
6
8
```

Grande! Sta eseguendo il suo compito in maniera perfetta. Oppure no? Proviamo con:

```
3
5
2
1
5
0
```

Questa volta otteniamo:

```
4      3
32155
4      3
21355
4      2
12355
4      1
12355
4      1
12355
4      1
12355
ecc.
```

Sta fornendo la risposta esatta, ma non esce mai dal ciclo. Notiamo che in questo caso c2 non arriva mai a zero e si può scommettere che è la condizione che concluderebbe il programma.

Che cosa determina che il programma debba entrare nel ciclo ridotto? La linea 80:

```
80  IF a (p) < a (p + 1) THEN GO TO 130
```

La differenza tra i due insiemi di dati è che il secondo contiene due valori identici. Siccome 5 non è inferiore a 5, si eseguirà il ciclo ridotto tutte le volte che si incontra la coppia di 5. Ecco perché il programma ripercorre continuamente il ciclo ridotto. Forse la questione può essere risolta con:

```
80  IF a (p) < = a (p + 1) THEN GO TO 130
```

Questa volta funziona tutto.

```
4      2
32155
4      2
21355
4      1
12355
4      0
12355
1
2
3
5
5
```

Ora il programma funziona meravigliosamente e possiamo togliere le linee di prova.

Spero di aver qui illustrato un paio di punti importanti. In primo luogo non abbiamo avuto bisogno di conoscere esattamente il funzionamento della procedura. Se avete lavorato attentamente fino a qui, ora è probabilmente tutto piuttosto chiaro; alcune esecuzioni simulate a mano probabilmente vi convincerebbero di aver capito. (Le esecuzioni manuali sono un ottimo metodo per capire il funzionamento di una procedura. Ne ho spesso fatte una dozzina su qualche oscuro pezzo di programma — naturalmente scritto da qualcun altro — prima che mi fosse veramente chiaro in mente che cosa facesse). Secondariamente, esiste sempre la tentazione di credere che, quando un programma funziona bene la prima volta, il lavoro è finito e c'è tempo per un bicchierino al bar dell'angolo. Come abbiamo visto, il lavoro *non* è finito, perché possono esistere altri insiemi di dati per i quali il programma fallisce; inoltre, comunque, se il tempo corre per voi come corre per me quando sto programmando, il bar ha chiuso un'ora e mezza fa.

17. Stringhe

I computer non sono solo dei mastica-numeri, possono anche digerire caratteri, cioè manipolare simboli

Arriva il postino... e c'è una lettera per voi. Una lettera molto personale. "Caro sig. Pigroni" dice, "Siete stato scelto tra un ristretto numero di abitanti di Borgorosso di Sotto per ricevere, assolutamente gratis, un meraviglioso paio di stivali Wellington foderati ...".

Hmm... Molto piacevole. Ma, la porta accanto, la vecchia signora Brontoloni ha ricevuto una lettera *quasi* identica. In effetti, a tutta Borgorosso di Sotto, insieme alla maggior parte degli abitanti della Pianura Padana, è successa la stessa cosa.

Ecco il listato del programma che "personalizza" le lettere.

```
  10 INPUT "Come vi chiamate ?";
n$
  20 INPUT "In che citta' risied
ete ?";c$
  30 INPUT "In che via abitate ?
";v$
  40 INPUT "A quale numero ?";n
  50 PRINT n$
  60 PRINT v$;", ";n
  70 PRINT c$
  80 PRINT "Caro sig ";n$;","
  90 PRINT "   Siete stato scelt
o tra un "
 100 PRINT "ristretto numero di
abitanti di"
 110 PRINT c$;" per ricevere "
 120 PRINT "assolutamente gratis
* un"
 130 PRINT "meraviglioso paio di
"
 140 PRINT "stivali Wellington f
oderati."
 160 PRINT "Siamo sicuri, caro s
ignor "
 170 PRINT n$;" che vorra' appro
fittare"
 180 PRINT "di questa vantaggios
a offerta,"
 190 PRINT "e che gli altri abit
anti di"
 200 PRINT c$;" La invidieranno
moltissimo."
 210 PRINT "   Vostro insincero,
"
```

```
 220 PRINT "      Milton F.Truffo
ni"
 230 PRINT "      Ca' dei Vendito
ri."
 240 PRINT ,,"* Spese postali Li
t 2.335.433 extra."
```

Fate eseguire questo programma e inserite (p. es.) "Giovanni Bianchi"; "Roma"; "Viale del Tramonto"; "666". Provate con altri nomi e indirizzi.

Ora immaginate che questo programma sia alimentato automaticamente con nomi e indirizzi da una banca dei dati, per sfornare migliaia di lettere ogni ora.

La cosa interessante è che non viene assolutamente richiesto del *calcolo*. Solamente memoria e qualche semplicissima manipolazione del testo scritto. Il computer può trattare una cosa del genere, dal momento che, esattamente come i numeri, può memorizzare delle *stringhe*, contrassegnate dal simbolo del dollaro, $.

Una *stringa* è una sequenza di *caratteri*. I caratteri sono elencati alle pag. 239-245 del *Manuale* e ciascuno ha un codice, di cui diremo tra un istante.

Ecco una stringa:

stringastringastringastringa.

Eccone un'altra:

ab334*./>>>><<—b+++++qqj.

(Se scrivete il punto fermo, anche questo fa parte della stringa!)

Una stringa può essere lunga un solo carattere, come <, o addirittura zero caratteri!

Per assegnare una variabile stringa, dovete porla tra virgolette:

10 LET a$ = "stringastringastringastringastringa"

Ogni variabile stringa deve avere un nome di una sola lettera seguito dal segno $.

Per una stringa lunga zero caratteri si usa LET a$ = " ".

Le stringhe obbediranno ai vostri comandi se saprete come porli. Un singolo carattere come

3

può essere considerato come

(a) un numero, 3
(b) una stringa, "3"

e potete passare dall'uno all'altra in vari modi. Supponiamo di considerarla davvero un stringa e di assegnarla:

10 LET a$ = "3"

Supponiamo di voler calcolare: 3 + 5. Non è buona cosa chiedere allo Spectrum di fare:

```
20  LET b = a$ + 5
30  PRINT b
```

Non funzionerà — provate se non mi credete. Perché no? È stato detto allo Spectrum che 3 va considerato una *stringa*, e non sa che questa è *anche* un numero. Comunque la possiamo *convertire* in numero usando VAL. Provate:

```
20  LET b = VAL a$ + 5
```

In generale, se un'espressione aritmetica è introdotta sotto forma di stringa, come

```
10  LET a$ = "2 + 2 + 3 * 5"
```

la macchina non sa che può essere calcolata aritmeticamente; si riferisce a questa solo come alla sequenza di caratteri:

2 + 2 + 3 * 5

Per esempio, potete estrarne il sesto carattere

```
20  PRINT a$ (6)
```

e ottenere la risposta: *. (Questo fatto può essere utile: come espressione aritmetica vale 19, che non *ha* un sesto carattere). Ma se *volete* convertirla in un numero, allora

```
20  PRINT VAL a$
```

fornirà la risposta: 19.

Si può convertire un numero in una stringa in due modi. Primo, inserendolo tra virgolette, per esempio: "3366". Ma se siete nel corso di un programma e state calcolando a + b, o qualsiasi altra espressione, che *vale* 3366, non si può scrivere "a + b" nella speranza di ottenere al suo posto "3366"; quello che otterreste è invece la stringa di tre caratteri:

a + b

CHR$ converte un numero compreso tra 0 e 255 in un singolo carattere, secondo la lista dei codici riportata alle pagg. 239-245 del *Manuale.* Per esempio CHR$96 è il segno di lira sterlina £. Alcuni numeri non sono usati.

CODE va in senso opposto: CODE "£" è 96. CODE "£ 335/h" è ancora 96: infatti CODE legge solo il primo carattere.

LEN vi dice quanto è lunga una stringa

La caratteristica di gran lunga più interessante delle stringhe è che

potete tagliarle in pezzi, detti *sottostringhe*. L'istruzione

```
a$ (3 TO 7)
```

estrae la sottostringa composta dal 3°, 4°, 5°, 6°, 7° carattere di a$. Dunque se a$ = "stringastringastringastringastringa", allora avremo a$ (3 TO 7) = "ringa". Si può naturalmente usare qualsiasi numero al posto di 3 e 7. Inoltre, a$ (5) estrae solo il 5° carattere — in questo caso "n".

Supponete, per esempio, di voler inserire un numero da 1 a 7 ed avere in uscita il giorno della settimana corrispondente (ponendo 1 per lunedì, 2 per martedì, ecc.). Si potrebbe scrivere

```
10  INPUT n
20  IF n = 1 THEN PRINT "Lun"
30  IF n = 2 THEN PRINT "Mar"
```

fino ad aver enumerato tutti e sette i giorni della settimana.

Invece, usando delle sottostringhe,

```
10  LET a$ = "LunMarMerGioVenSabDom"
20  INPUT n
30  PRINT a$ (3 * n — 2 TO 3 * n)
```

si ottiene lo stesso risultato con 5 linee in meno.

Potete collegare le stringhe l'una di seguito all'altra usando + per unirle, come segue

```
"cara" + "mella" = "caramella"
```

oppure disporle in ordine alfabetico usando <. Consultate il *Manuale* e fate qualche esperimento. Un attento uso delle stringhe può spesso far risparmiare un sacco di spazio.

Questo programma accetta un nome e ne trova le iniziali.

```
10  INPUT "Dimmi il tuo nome e cognome"; n$
20  LET n$ = "□" + n$
30  FOR i = 1 TO LEN n$
40  IF n$ (i) = "□" AND i < LEN n$ THEN PRINT n$ (i + 1); ".";
50  NEXT i
```

Eseguitelo usando il vostro nome. Inserite "Compagnia Generale Semiconduttori" e controllate se risponde "C.G.S.".

Così come è, non è perfetto: se ponete spazi supplementari tra le parole, produce una uscita piuttosto confusa. Tutto quello che fa è apporre uno spazio extra all'inizio della stringa, per assumere poi che qualsiasi carattere seguente uno spazio è una iniziale. Per chiarezza inserisce dei punti.

Progetto

Modificate il programma in modo che ignori gli spazi ripetuti. Un metodo è di compiere una ricerca, per eliminare ogni spazio successivo a uno dato. Per cancellare l'ennesimo carattere dalla stringa n$, potete scrivere:

```
LET n$ = n$ (TO n — 1) + n$ (n + 1 TO)
```

Se si omettono i numeri prima o dopo "TO", la macchina li pone rispettivamente uguali all'estremità iniziale e a quella finale.

18. Data

Un metodo più efficiente per assegnare valori alle variabili

Al di sopra del tasto D, in verde, c'è la parola ''DATA''. Usando questa istruzione si può evitare di dover battere lunghe serie di istruzioni del tipo LET a(37) = 242, LET a(38) = 243,...ecc. Per esempio, e solamente per familiarizzarvi con il comando, provate a scrivere ciò che segue:

```
10  DATA 1, 2, 3, 4, 5, 6, 7
20  FOR i = 1 TO 7
30  READ x
40  PRINT x
50  NEXT i
```

Come uscita dovreste ottenere ancora i numeri 1-7. Essenzialmente il comando READx significa ''LET x = il prossimo numero della lista DATA''. Ogniqualvolta incontra una istruzione READ, il computer cerca lungo le linee DATA, trova l'ultimo elemento letto, e assegna il successivo come valore della variabile che segue l'istruzione READ. Dunque la prima volta legge x come 1, la seguente come 2, e così via.

Una caratteristica utile di DATA è che la macchina considera tutte le linee DATA di un singolo programma come una lista unica, collegandole l'una di seguito all'altra. Potete disporre queste linee dove preferite (benché, dal momento che il computer le deve ricercare, un luogo prossimo all'inizio tenda ad essere più rapido). Provate a riscrivere il programma, per esempio in questa forma:

```
10  FOR i = 1 TO 7
20  READ x
30  DATA 1, 2, 3, 4
40  PRINT x
50  DATA 5
```

```
60  NEXT i
70  DATA 6, 7
```

Funziona proprio nello stesso modo — perfino se la linea DATA è all'interno di un ciclo FOR/NEXT (DATA costituisce un'eccezione alla regola che vuole le linee eseguite in ordine numerico).

Se questo fosse tutto quello che potete fare, non sarebbe molto impressionante. Ma, ecco un utilizzo più tipico.

```
10  DATA 100, 50, 150, 100, 100, 150, 50, 100, 100, 50
20  READ x, y
30  PLOT x, y
40  FOR i = 1 TO 4
50  LET x0 = x: LET y0 = y
60  READ x, y
70  DRAW x — x0, y — y0
80  NEXT i
```

Questo dovrebbe disegnare una figura a forma di diamante: la lista DATA dà le coordinate degli angoli, con l'angolo inferiore ripetuto sia all'inizio che alla fine.

Sviluppiamo l'idea con un gatto...

```
10  DATA 2, 0, 6, 0, 9, 1, 15, 0, 16, 1, 16, 12, 15, 13, 14, 12,
    14, 2, 10, 2, 11, 6, 8, 12, 10, 15, 9, 18, 8, 22, 7, 18, 3, 18, 2,
    22, 1, 18, 0, 15, 2, 12, 0, 6, 0, 3, 2, 0
20  READ x, y
30  PLOT x, y
40  FOR i = 1 TO 23
50  LET x0 = x: LET y0 = y
60  READ x, y
70  DRAW x — x0, y — y0
80  NEXT i
```

Analizzando quello che fa, scoprirete che prima disegna (con PLOT) il punto di coordinate 2, 0 per poi congiungerlo al punto 6, 0; poi 9, 1; e così via. I numeri sono quelli che si susseguono nella lista DATA.

Per posizionare la figura più verso il centro dello schermo, trasformate la linea 30 in:

```
30  PLOT 100 + x, 60 + y
```

La lista DATA è stata calcolata facendo un disegno approssimativo su carta millimetrata e leggendo le coordinate. È proprio come le

figure in quei libri di disegni da punto a punto, con l'eccezione che noi dobbiamo specificare dove si trovano i punti tramite le loro coordinate.

Ora, c'è ancora un sacco di lavoro solo per un gatto. Per la scelta di gatti grossi, magri, e persino a gambe all'aria o che guardano in un'altra direzione, potete *modificare* i dati. Sostituite la linea 30 come già detto, per fare spazio; aggiungete questa linea

```
 5   INPUT a, b
```

e trasformate la linea 70 in

```
70   DRAW a * (x — x0), b * (y — y0)
```

Ora, in esecuzione, dovete introdurre due numeri. All'inizio non siate eccessivamente ambiziosi: provate a = 1, b = 2; anche a = 2, b = 1. Provate quindi a = 2, b = —1; a = —2, b = —1. Poi provate quello che volete! Ma fate attenzione: il programma non è protetto contro la possibilità di andare fuori dallo schermo, e valori frazionari di a e b danno risultati divertenti a causa degli errori di arrotondamento. Per un trucco in grado di evitare questo problema, vedi il Programma preconfezionato sulle spirali.

Disegnamo ora un'intera linea di gatti. Allo scopo, possiamo usare un ciclo; ma ci serve un modo per posizionare nuovamente l'istruzione READ all'inizio della lista dei dati. Questo è proprio quello che fa l'istruzione RESTORE. Aggiungiamo perciò le linee

```
 15   FOR t = 1 TO 8
 90   RESTORE
100   NEXT t
```

e trasformiamo la linea 30 in

```
30   PLOT 50 + x + 20 * t, 50 + y
```

È tutto.

Progetti

1. Fate sì che i gatti si posizionano in diagonale sullo schermo, da sinistra in basso verso destra in alto, come se sedessero su una rampa di scale.
2. Aggiungete altre DATA per disegnare le scale.
3. Sostituite la lista DATA, aiutandovi con un atlante, in modo che il programma disegni una carta dell'Australia. Scoprite come si presenta l'Australia capovolta (gli australiani pensano che lo sia).
4. Se avete mezzo pomeriggio libero, stabilite le DATA per un mappamondo.

19. Debugging IV

Funziona davvero il programma?

Come possiamo definitivamente dimostrare che un programma fa esattamente quello per cui era stato scritto? Non voglio rimanere coinvolto in una discussione filosofica troppo complessa ma, a grandi linee, è un po' come chiedere a un astronomo se domani sorgerà il Sole. Se fosse molto pedante, potrebbe rispondere che la Terra ha continuato a ruotare intorno al Sole per molto tempo e che abbiamo un corpo di leggi fisiche che suggerisce che continuerà a farlo in modo regolare, e che si potrebbe scommettere che anche domani continuerà ad essere così; tuttavia aggiungerebbe che non ha modo di sapere se le nostre leggi fisiche sono esatte e che quello che abbiamo osservato per migliaia di anni, potrebbe in effetti essere una manifestazione di una legge molto più complessa, il cui effetto potrebbe essere, domani, di invertire la direzione della rotazione della Terra oppure di toglierla completamente dall'orbita.

Analogamente, il fatto che un programma si comporti correttamente per il primo migliaio di insiemi di dati in ingresso, non è una garanzia assoluta del fatto che funzionerà per il milleunesimo. Infatti i bachi spesso non appaiono per mesi o persino anni dopo che il programma è stato felicemente portato a termine e che ha girato senza problemi decine o addirittura centinaia di volte. Tutto ciò non deve sorprendere; dopo tutto, sono le condizioni che si verificano più raramente quelle che il programmatore è più probabile trascuri.

Ecco un esempio: stiamo scrivendo una serie di programmi per la Compagnia Elettrica Nether Hopping per la gestione della contabilità con i clienti. Ci spiegano che esistono due tariffe, A e B. Con la tariffa A l'utente paga una quota fissa trimestrale di 40 000 lire, oltre al consumo al prezzo unitario di 150 lire. Con la tariffa B il consumatore non paga la quota fissa, ma paga il consumo a 230 lire. Scriviamo così uno spezzone di programma come:

```
100  INPUT t$
105  INPUT consumo
110  IF t$ = "a" THEN GO TO 300
```

```
 120  IF t$ = "b" THEN GO TO 140
 130  GO TO 5000
 140  LET bolletta = 230 * consumo
 150  PRINT bolletta
 160  GO TO 100
 300  LET bolletta = 40000 + 150 * consumo
 310  PRINT bolletta
 330  GO TO 100
      .....
5000  PRINT "Codice tariffa inesistente"
5010  STOP
```

OK. Lo so che il programma potrebbe essere più efficiente e che in realtà ci servirebbe qualche altra informazione, come il nome dell'utente e il numero del conto, ma questa è l'idea fondamentale.

Proviamo dunque questa parte di programma — funziona bene, così ce ne andiamo borbottando che è uno spreco del nostro notevole talento che ci vengano dati da scrivere programmi sciocchi come questo.

E funziona bene; per anni; e un giorno stampa una bolletta per 0.00 lire. Naturalmente nessuno la nota, siccome è una delle migliaia di bollette e, in ogni caso, viene probabilmente imbustata in modo automatico. Il destinatario è confuso e forse divertito dalla bolletta, perché dimostra quanto siano stupidi i computer, ma non sembra esserci alcun motivo di fare qualche cosa, così butta via la bolletta. Sfortunatamente, nella stessa serie abbiamo scritto un programma che memorizza la data di spedizione di ogni bolletta e spedisce un avviso se non riceve conferma che la bolletta è stata pagata entro 28 giorni. Questa volta il destinatario è più irritato che divertito, ma la affida solo al cestino della carta straccia, come prima. A questo punto le cose iniziano ad andare notevolmente male. Il programma che controlla il ritardo tra l'emissione del conto e il suo pagamento emette un ordine per la squadra di manutenzione perché isoli l'utente se dopo 60 giorni non ha ancora pagato.

Che cosa era successo? Facile!! L'utente è un pensionato che ha approfittato di uno dei lunghi viaggi organizzati per le vacanze invernali che le agenzie di viaggio offrono agli anziani. È stato fuori dal paese per appena più di tre mesi e non ha usato elettricità per tutto un periodo di fatturazione. Inoltre è un'utente di elettricità insolitamente parco e quindi è inserito nella tariffa B. Ecco perché il sistema ha emesso una richiesta per un pagamento nullo, il che naturalmente non succederà molto spesso, dato che poca gente sta lontana da casa per così tanto tempo ed è anche probabile che gli utenti a tariffa B siano molto pochi. Perché il problema si presenti l'utente deve soddisfare entrambi i requisiti.

Una volta individuato, il baco può essere schiacciato:

```
145  IF bolletta = 0 THEN GO TO 100
```

Così si evita l'istruzione PRINT. Si dice che questo problema si sia effettivamente presentato in uno dei primi sistemi computerizzati, per quanto non saprei dire se si tratti di una favola o meno. In ogni caso penso che illustri chiaramente come un baco possa dormire praticamente all'infinito.

La morale è: quando inventate dei dati per provare un programma, non fatelo a caso. Scegliete dei valori prossimi a quelli di decisione nel programma. Se un'istruzione dice:

```
305  IF u < 30 THEN GO TO 500
```

eseguite una prova con u posto a 29.999 e una a 30.0001. Può darsi che intendeste:

```
305  IF u < = 30 THEN GO TO 500
```

Se provate solo u = 15 e u = 160 non noterete l'errore.

Assicuratevi che i dati di prova siano stati scelti in modo che ogni sezione del programma venga eseguita prima o poi. E naturalmente assicuratevi di conoscere esattamente quale dovrebbe essere la risposta giusta per ogni insieme di dati di prova.

20. Curve

Solo per menti matematiche!

Abbiamo già incontrato, ma solo incidentalmente, il concetto di *grafico* determinato da coordinate. Riepiloghiamo. Iniziamo con due linee, l'asse x e l'asse y, poste ad angolo retto l'una rispetto l'altra. Lungo queste possiamo definire delle distanze x e y (segnando i numeri negativi alla sinistra sull'asse x e verso il basso sull'asse y) e usare queste due distanze per determinare un punto di *coordinate* x e y. Proprio come per i pixel. (Vedi Figura 20.1).

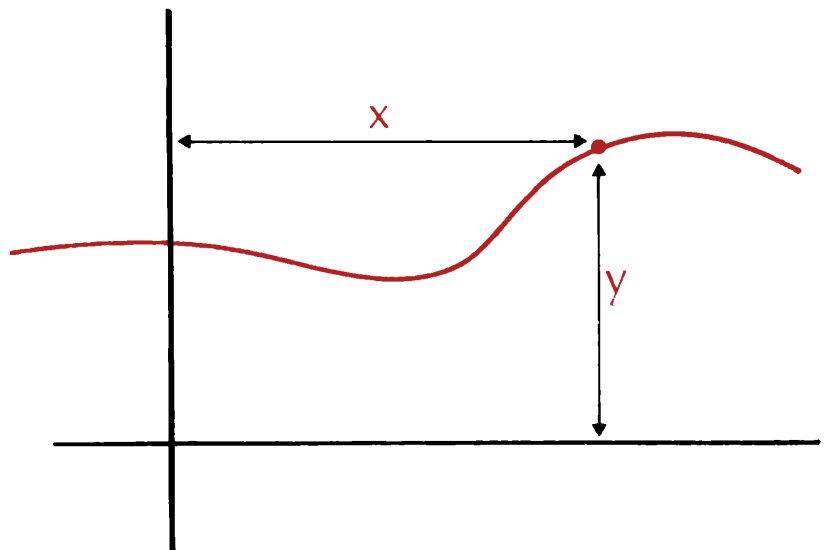



Figura 20.1 - Coordinate per il tracciamento di curve.

Se ora immaginiamo x variabile lungo l'asse x e il valore di y che cambia in qualche modo dipendente da x, allora anche il punto di coordinate (x, y) si muoverà; e, in generale, traccerà una *curva*. Una formula che descriva come y dipenda da x, diciamo $y = x^2 - 3$, determina quindi la forma di una particolare curva. (Questa idea, dovuta a Cartesio nel 1637, permette di studiare curve geometriche tramite l'algebra: è il punto di partenza dell'analisi).

Tramite l'istruzione PLOT possiamo disegnare curve simili:

```
10  FOR j = 0 TO 255
20  PLOT j, j/2
30  NEXT j
```

Dovreste ottenere una linea di pixel che salgono lungo lo schermo dall'angolo inferiore sinistro a quello superiore destro. Un matematico la chiamerebbe il grafico della funzione y = x/2.

Potete modificarla per fornire una infinita varietà di grafici, semplicemente sostituendo j/2 con qualche altra espressione collegata a j. Per esempio, per tracciare la radice quadrata di j in funzione di j, basta trasformare j/2 in SQR j, ottenendo:

```
10  FOR j = 0 TO 255
20  PLOT j, SQR j
30  NEXT j
```

È un poco più interessante, vero?

Fate esperimenti. Trasformate ancora j/2. Provate queste.

```
(Parabola)      20  PLOT j, .002 * j * j
(Sinusoide)     20  PLOT j, 80 * SIN (j/20) + 80
(Cosinusoide)   20  PLOT j, 80 * COS (j/20) + 80
(Catenaria)     20  PLOT j, (EXP (.02 * (j — 120))
                    + EXP (.02 * (120 — j))) * 10
```

...perseverate. Perché sono così complesse?

Sorgono dei problemi se semplicemente scrivete al posto di j/2 una qualsiasi espressione nota. La questione è legata a quello che entrerà nello schermo. Il valore della coordinata verticale deve essere compreso tra 0 e 175, altrimenti lo Spectrum perde la pazienza e chiude bottega. (Il poverino non può disegnare nient'altro e si risente se gli si richiede di farlo.) Avete perciò bisogno di regolare la *scala* per fare sì che le cose ci stiano. Ne capirete immediatamente il perché se provate con:

```
20  PLOT j, j * j
20  PLOT j, SIN j
20  PLOT j, COS j
20  PLOT j, EXP (j) + EXP (—j)
```

Si può aggirare il problema, naturalmente — vedere più oltre, al paragrafo *Scala*. Prima, però, ecco alcune altre funzioni interessanti che effettivamente stanno nello schermo, dal momento che le ho scelte molto accuratamente allo scopo.

```
20  PLOT j, EXP (—j/80) * SIN (j/8) * 80 + 80
```

```
20  PLOT j, 80 + 80 * COS SQR (2 * j)
20  PLOT j, ABS (j — 127)
20  PLOT j 80 + 60 * LN (1 + ABS SIN (j * .125))
20  PLOT j, 12 * ABS (j/4 — 30) ↑ .666
20  PLOT j, 40 * ABS (j/4 — 30) ↑ .25
20  PLOT j, 160 * EXP (— .1 * (j/4 — 30) * (j/4 — 30))
```

Per evitare di scrivere espressioni complicate potete costruire la formula in stadi successivi, così:

```
20  LET t = j/24
25  PLOT j, 120 + .1 * t * (t — 2) * (t — 4) * (t — 6) * (t — 8) * (t — 10)
```

Scala

Momentaneamente ci occuperemo solo di funzioni definite per numeri positivi e che assumano valori positivi: una eccellente da usare è SQR, la radice quadrata. Prendete il precedente programma per il tracciamento di grafici e sostituite di volta in volta la linea 20 con una delle seguenti:

```
(a)   20  PLOT j, SQR j
(b)   20  PLOT j, 2 * SQR j
(c)   20  PLOT j, 4 * SQR j
(d)   20  PLOT j, 6 * SQR j
(e)   20  PLOT j, 8 * SQR j
(f)   20  PLOT j, 10 * SQR j
(g)   20  PLOT j, 12 * SQR j
(h)   20  PLOT j, 14 * SQR j
```

Noterete abbastanza rapidamente che in grafici successivi ogni punto va in posizioni più alte dello schermo — in effetti, in (g) la macchina si ferma perché i punti del grafico iniziano a uscire dallo schermo. Più grande diventa il "qualche cosa" nella espressione (qualche cosa) * SQR j, più il grafico viene stirato nella direzione verticale. Questo "qualche cosa" è un *fattore di scala*, e modificandolo potete fare sì che i grafici entrino esattamente nello schermo.

Se il fattore di scala è troppo piccolo, si ottengono grafici così schiacciati da non poter distinguere i particolari. Provate:

```
20  PLOT j, .1 * SQR j
```

Se la funzione che state disegnando cresce troppo rapidamente, potete riportarla nello schermo regolando il fattore di scala. Per esempio

```
20  PLOT j, j * j
```

esce dallo schermo perché 14 * 14 = 194, che è già troppo grande.

In effetti il numero più grande sarà 255 * 255 = 65025. Dividendolo per 400 ottenete 162.5625, che è minore di 175, il valore massimo permesso. Posto quindi di utilizzare 1/400 come fattore di scala, si ottiene un buon grafico:

```
20  PLOT j, j * j/400
```

Esiste una regola generale piuttosto ovvia che assicura di avere scelto correttamente il fattore di scala. Supponiamo che sia m (massimo) il valore più elevato assunto dalla funzione al variare di j tra 0 e 255. Detto f il fattore di scala, ogni cosa andrà a posto se si fa in modo che f * m sia minore o uguale a 175, e possibilmente abbastanza vicino a questo valore, in modo da non avere un grafico troppo schiacciato. In particolare, potete ottenere f * m = 175, ponendo f = 175/m. (Per figure tonde può essere più adeguato 160/m; inoltre di m serve una stima ragionevole, più che il valore esatto).

In effetti potreste scrivere un programma per calcolare m. Supponete di applicarlo alla funzione j * j; questo farà il giochetto:

```
10  LET m = 0
20  FOR j = 0 TO 255
30  LET q = j * j
40  IF q > m THEN LET m = q
50  NEXT j
```

Così com'è, però, non disegnerà il grafico; aggiungete la routine di tracciamento che segue:

```
20  FOR j = 0 TO 255
70  PLOT j, (175/m) * j * j
80  NEXT j
```

Un difetto: dovete eseguire tutti i calcoli *due volte*, e ciò è difficile da evitare in modo efficiente, a meno che non sappiate *quale* j dia il valore massimo di j * j. In questo caso è ovviamente j = 255, ma non sempre è così facile da vedere in anticipo. (*Potete* dimensionare un vettore v (i) di dimensione 256, memorizzare i valori di j * j sotto forma di v(j + 1), per poi usare questi per PLOT; tuttavia i vettori e le matrici consumano un sacco di memoria! Consultate il *Manuale* per ulteriori dettagli su vettori e matrici).

Esattamente come potete modificare la scala sull'asse verticale, potete farlo sull'asse orizzontale. Le funzioni SQR j o j * j non mostrano questa necessità molto chiaramente, così userò 80 + 80 * SIN j. Provate i seguenti esempi:

```
20  PLOT j, 80 + 80 * SIN (.05 * j)
20  PLOT j, 80 + 80 * SIN (.1 * j)
20  PLOT j, 80 + 80 * SIN (.15 * j)
```

```
20  PLOT j, 80 + 80 * SIN (.2 * j)
20  PLOT j, 80 + 80 * SIN (.25 * j)
```

Questa volta cambia la scala *orizzontale* — ma il fattore di scala opera in modo piuttosto diverso (avete notato?). Tanto maggiore è il fattore di scala, ora, tanto più il grafico viene schiacciato in direzione orizzontale: ottenete più ondeggiamenti della curva. Perché?

Quando j spazia da 0 a 255, il numero .05 * j varia da .05 * 0 a .05 * 255, cioè da 0 a 12.75.

Quando j va da 0 a 255, .1 * j varia da .1 * 0 a .1 * 255, cioè da 0 a 25.5.

Dunque, nel secondo caso, si comprime nello stesso spazio orizzontale un numero *doppio* di valori.

Infatti, con un fattore di scala f, cioè usando

```
20  PLOT j, 80 + 80 * SIN (f * j)
```

disegnate il campo da 0 a f * 255 nella larghezza dello schermo. Tanto maggiore è f, tanto più vasto è il campo esplorato, tanto più schiacciato diventa il disegno.

Dunque, volendo disegnare per una data estensione, per esempio 1000, desiderate che f * 255 = 1000, cioè f = 1000/255. In generale, se vi interessa il campo da 0 a n, vi servirà il fattore di scala n/255.

Riassumendo:

$$\text{migliore fattore di scala verticale} = \frac{175}{\text{valore più grande da disegnare}}$$

$$\text{migliore fattore di scala orizzontale} = \frac{\text{valore massimo della variabile}}{255}$$

Progetto

Se non sapete che cosa sia meglio, potete scrivere un programma "interattivo" che vi permetta di scegliere i due fattori di scala (tramite INPUT) per poi disegnare il grafico; se il risultato non vi piace, lo eseguite ancora e cambiate la scala.

Scrivete un programma simile per la funzione 80 + 80 * SIN j.

Suggerimento: se fo è il fattore di scala orizzontale e fv quello verticale, la linea di programma operativa sarà:

```
20  PLOT j, fv * (80 + 80 * SIN (fo * j))
```

Traslazioni degli assi

Potreste domandarvi: "Perché tutti quegli 80 + 80 * SIN j?". Oppure: "Molto bene, ma se i numeri fossero negativi?". La risposta è identica in entrambi i casi. Il modo di trattare i numeri negativi consiste nello spostare gli assi.

Provate questo programma.

```
10  INPUT s
20  FOR j = 0 TO 255
```

```
30  PLOT j, s + SQR j
40  NEXT j
```

Provate a introdurre s = 0, 10, 20, ecc.

Quello che vedrete è lo stesso grafico, ma che si muove verso l'alto sullo schermo in funzione del valore di s. Ci sono due modi di considerare la cosa.

Il primo è che state disegnando funzioni differenti, 5 + SQR j, oppure 10 + SQR j, ecc.

L'altro è che state sempre disegnando SQR j, ma che cambia la posizione dell'asse x sullo schermo. Guardate la Figura 20.2 che si spiega molto bene da sola.

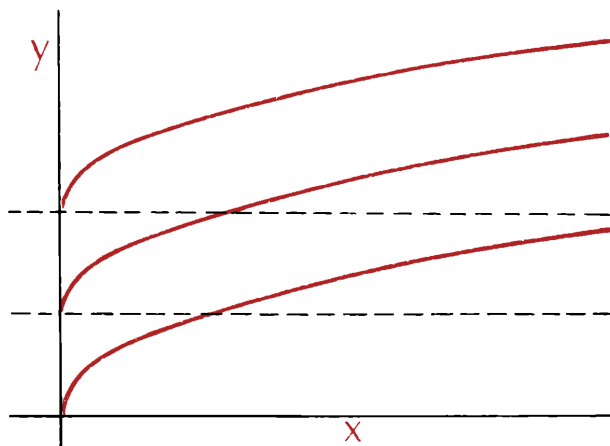



Figura 20.2 - Sommare una costante equivale a spostare l'asse x.

Per ottenere una chiara sinusoide, per esempio, dovete sostituire la precedente SQR j con 80 * SIN (.1 * j) e scegliere s in modo da avere tutto in posizione centrale sullo schermo. Un buon funzionamento si ha con s = 80; da cui tutti quegli 80 + 80 * SIN j.

Questo per far muovere l'asse x. Per spostare l'asse y, potete modificare l'escursione di j, per esempio da —120 a 135 invece che da 0 a 255.

Potete combinare questi spostamenti con variazioni di scala in entrambe le direzioni, e non è necessario mantenere gli assi centrali — dipende da voi. Mi è capitato di scrivere una specifica dettagliata per ottenere sullo schermo il miglior quadro possibile di una data curva in un dato intervallo; ho poi dato uno sguardo alla massa di algebra che avevo scritto, e ho buttato via il tutto. A volte la matematica nasconde il bosco dietro gli alberi.

Personalmente vi suggerisco di fare invece qualche prova, usando il seguente programma:

```
10  INPUT a, b, c, d
20  PLOT 0, d
30  DRAW 255, 0
40  LET u = —b/a
```

```
50  PLOT u, 0
60  DRAW 0, 175
100 FOR j = 0 TO 255
110 PLOT j, c * SIN (a * j + b) + d
120 NEXT j
```

In questo caso a, b, c, d spostano assi e scala mentre l'asse x e l'asse y vengono disegnati. Non ho aggiunto protezioni contro la possibilità che la funzione esca dai valori permessi — pigrizia a parte, questo conduce alla necessità di porre attenzione nella scelta delle scale e delle posizioni degli assi.

(Provate a = .1, b = —10, c = 80, d = 80. Il valore negativo di b è generalmente necessario).

Altre tecniche

Ci sono altri due modi di usare PLOT e DRAW per ottenere delle curve. Invece di sovraccaricare questo capitolo di tecnicismi, ne ho inseriti degli esempi nei Programmi preconfezionati. Vedi *Figure di Lissajous, Spirali e rosette, Dimostrazione grafica 1* e *2*.

21. Debugging V

Alle volte dei numeri che sembrano uguali... non lo sono!

Finora i tipi di bachi di cui ci siamo occupati erano prodotti da noi stessi, ed erano ragionevolmente facili da eliminare, una volta trovati. Esiste un altro tipo di bachi generati dalle caratteristiche stesse della macchina. Non si tratta di un errore di progetto, ma di una conseguenza del modo in cui sono organizzati tutti i computer. Ha a che fare con la precisione con cui i calcolatori memorizzano i numeri. Se pensiamo a uno qualsiasi dei modi comuni di conservare i numeri, è ovvio che esiste un limite al numero di cifre utilizzabili. Un contachilometri di una automobile per esempio, può tenere solo 5 cifre, dato che ha solo 5 ''finestrelle''. Con un computer è lo stesso. Ogni numero può occupare non più di un dato numero di ''finestrelle''. Tuttavia, ogni finestrella non rappresenta una cifra decimale. Il codice macchina interno relativo ai numeri è piuttosto diverso dal modo in cui noi li pensiamo e non vi annoierò con i dettagli. Il fatto che c'è una intrinseca approssimazione *e* che si usa una conversione di codice, significa che la rappresentazione esterna di un numero (come viene mostrato sullo schermo) può non essere proprio la stessa cosa di quella interna. Vi farò un esempio di ciò di cui sto parlando tratto degli esercizi scolastici. Se moltiplicate 2 per 2 usando i logaritmi avete:

No.	Log.
2	0.3010
2	0.3010
3.999	0.6020

cioè 2 × 2 = 3.999!

La combinazione del fatto che abbiamo utilizzato logaritmi con solo 4 cifre decimali (cioè che possono occupare solo 4 caselle) e che ha luogo una conversione di codice (dal numero al logaritmo, dal logaritmo di nuovo al numero) crea l'imprecisione.

Ecco un programma che causa lo stesso tipo di problema:

```
10  FOR p = 0 TO .3 STEP .01
20  LET q = ATN (TAN (p))
30  IF p <> q THEN PRINT p, q
40  NEXT p
```

Alla linea 20 otteniamo la tangente di p e immediatamente invertiamo il procedimento calcolando l'arcotangente del risultato. In altre parole, q dovrebbe contenere gli stessi valori di p. Così la linea 30 non dovrebbe mai avere l'effetto di stampare p e q, dato che dovrebbero essere sempre uguali. Quando facciamo girare il programma, otteniamo la seguente uscita:

```
0.02    0.02
0.03    0.03
0.04    0.04
0.05    0.05
0.07    0.07
0.09    0.09
0.11    0.11
0.12    0.12
0.13    0.13
0.14    0.14
0.16    0.16
0.18    0.18
0.2     0.2
0.21    0.21
0.22    0.22
0.26    0.26
0.28    0.28
```

In verità, questo è un risultato molto strano perché la macchina non solo sta stampando dei valori, con ciò affermando che sono differenti, ma li sta anche stampando come se fossero uguali! Quello che è accaduto è che il complesso procedimento matematico implicato ha portato a delle piccole imprecisioni nella rappresentazione interna dei numeri a cui vanno imputate le differenze tra p e q. Tuttavia sono implicate delle imprecisioni anche nel decodificare il formato interno nei numeri decimali mostrati sullo schermo, così questi appaiono identici benché la macchina con cuore di pietra affermi il contrario. Si noti che per alcuni valori i codici interni *sono* uguali: per esempio per 0.06 e 0.08.

Questo tipo di errori può essere estremamente sconcertante e alle

volte l'unica via d'uscita è consentire un piccolo margine d'errore per mezzo dell'istruzione IF, in modo da avere:

IF ABS (p — q) < 0.000001 THEN...

La funzione ABS è necessaria perché q può essere maggiore di p. Per esempio, se p = 3 e q = 3.1, allora p — q = — 0.1, che è minore di 0.000001, così la condizione sarebbe soddisfatta se non ci fosse la funzione ABS (che in pratica elimina il segno meno). Invece calcolando ABS (—0.1) = 0.1, che è maggiore di 0.000001, la condizione non è soddisfatta, come volevamo.

22. Stile di programmazione

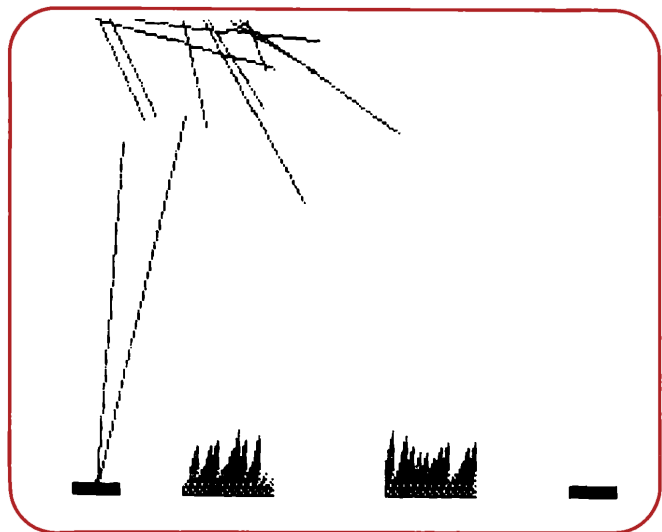
Scrivere programmi lunghi e complessi è anche una questione di stile

Cominciamo dal principio: il programma. Se volete potete considerarlo un programma preconfezionato: copiatelo e fatelo girare.

Inizia disegnando due città e due postazioni antimissile chiamate silo "1" e "2" da sinistra a destra.

Ora piovono su di voi dei missili in modo abbastanza imprevedibile, ma la traccia di un dato missile ha un angolo fisso.

Premete "1" o "2" sulla tastiera per attivare uno dei vostri silo difensivi. (Il programma può avere bisogno di un poco di tempo per rispondere, perciò tenete premuto il tasto finché non lo fa). Vi si domanderà poi una gittata e un alzo per il missile d'intercettazione. Il fatto che la relativa testata cessa di lampeggiare, vi indica che avete colpito uno dei missili in arrivo.

Prima o poi finirete i vostri missili (ce ne sono 20 in ogni silo, all'inizio), oppure saranno distrutte le vostre città o le vostre postazioni. Scopo del gioco è di abbattere quanti più missili vi è possibile, prima che questo accada.

È necessaria qualche spiegazione sui valori di "gittata" e "alzo" che dovete introdurre.

In primo luogo, il valore della gittata usa lo stesso sistema di coordinate dello Spectrum nel modo grafico ad alta risoluzione (PLOT). Così, se volete colpire qualcosa nell'estremità alta dello schermo immediatamente sopra un silo, la gittata è 175. Dall'angolo inferiore sinistro a quello superiore destro, la distanza totale è circa 300.

Secondo, l'alzo definisce l'angolo di tiro rispetto all'orizzonte alla destra del silo a cui ci si riferisce. Questo significa che per il silo 2 gli angoli più utili si trovano tra 90° e 180°, dunque, attenzione! (Per il silo 1, comunque, i valori utili vanno da 0° a 90°). Una delle caratteristiche del programma è che non vi dice se commettete un errore come spedire un missile fuori dallo schermo. Toglierà semplicemente un missile dal vostro stock e non ne disegnerà alcuna traccia.

Ecco il listato completo.

Battaglia missilistica

```
   1 RANDOMIZE
  10 LET disegno=500: LET genmis
=1000
  20 LET movmis=1500
  30 LET fuoco=2500: LET prnscr=
3000: LET contcolp=3500
  40 DIM m(3,20): DIM s(2)
  50 LET citt=2: LET s(1)=20: LE
T s(2)=20
  60 LET colsilo=4000: LET colci
t=4500
  70 LET punt=0
  80 LET pretas=5000
 100 GO SUB disegno
 110 IF citt<1 OR s(1)+s(2)=0 TH
EN GO SUB prnscr
 120 GO SUB genmis
 130 GO SUB movmis
 160 GO TO 110
 500 LET xs=8
 510 FOR l=1 TO 2
 520 FOR x=xs TO xs+20
 530 PLOT x,0
 540 DRAW 0,4
 550 NEXT x
 560 LET xs=224
 570 NEXT l
 580 LET xs=56
 590 FOR l=1 TO 2
 600 FOR x=xs TO xs+36 STEP 3
 610 LET h=INT (RND*10)+3
 620 FOR y=0 TO h
 630 PLOT x,y
 640 DRAW 3,y
 650 NEXT y
 660 NEXT x
 670 LET xs=144
 680 NEXT l
 690 RETURN
1000 FOR l=1 TO 5
1010 LET x=INT (RND*100)
1020 LET a=(RND*PI/2)+.01
1030 FOR p=1 TO 20
1040 IF m(3,p)=0 THEN LET m(1,p)
=x: LET m(2,p)=175: LET m(3,p)=a
: GO TO 1060
1050 NEXT p
1055 GO SUB pretas
```

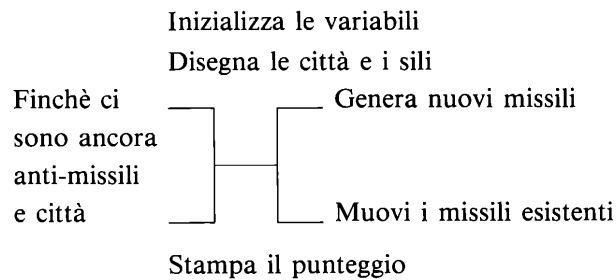
```
1060 NEXT l
1070 GO SUB pretas
1080 RETURN
1500 FOR p=1 TO 20
1505 IF m(3,p)=0 THEN GO TO 1580
1510 PLOT FLASH 0;m(1,p),m(2,p)
1520 LET xo=20*COS m(3,p)
1530 LET yo=-20*SIN m(3,p)
1540 IF m(1,p)+xo>255 OR m(2,p)+
yo<0 THEN LET m(3,p)=0: GO SUB c
ontcol?: GO TO 1580
1550 DRAW xo,yo
1560 LET m(1,p)=m(1,p)+xo
1570 LET m(2,p)=m(2,p)+yo
1573 PLOT FLASH 1;m(1,p),m(2,p)
1575 GO SUB pretas
1580 NEXT p
1585 GO SUB pretas
1590 RETURN
2500 INPUT "gittata,alzo";gi,al
2505 IF s(VAL d$)=0 THEN RETURN
2510 IF d$="1" THEN LET xb=16
2520 IF d$="2" THEN LET xb=234
2523 PLOT xb,0
2525 LET s(VAL d$)=s(VAL d$)-1
2530 LET xf=gi*COS (al*PI/180)
2540 LET yf=gi*SIN (al*PI/180)
2545 IF xf+xb<0 OR xf+xb>255 OR
yf<0 OR yf>175 THEN RETURN
2550 DRAW xf,yf
2560 FOR p=1 TO 20
2570 IF ABS (xf+xb-m(1,p))>15 OR
 ABS (yf-m(2,p))>15 THEN GO TO 2
700
2610 LET punt=punt+1
2625 PLOT FLASH 0;m(1,p),m(2,p)
2630 LET m(3,p)=0
2700 NEXT p
2710 RETURN
3000 CLS
3010 IF citt<1 THEN PRINT AT 10,
2;"citta' distrutte"
3020 IF s(1)+s(2)=0 THEN PRINT A
T 10,2;"anti-missili terminati"
3030 PRINT AT 12,2;punt;" missil
i abbattuti"
3040 GO TO 9999
3500 LET xt=m(1,p)+xo
3510 IF xt>7 AND xt<29 THEN LET
xs=8: GO SUB colsilo
3520 IF xt>223 AND xt<245 THEN L
ET xs=224: GO SUB colsilo
3530 IF xt>55 AND xt<93 THEN LET
 xs=56: GO SUB colcit
3540 IF xt>143 AND xt<181 THEN L
ET xs=144: GO SUB colcit
3550 RETURN
4000 IF xs=8 THEN LET s(1)=0
4019 IF xs=224 THEN LET s(2)=0
4020 PRINT AT 21,xs/8;"    "
4030 RETURN
4500 LET citt=citt-1
4510 FOR r=19 TO 21
4520 PRINT AT r,xs/8;"
4530 NEXT r
4540 RETURN
5000 IF INKEY$="" THEN RETURN
5010 LET d$=INKEY$
5015 IF CODE d$<49 OR CODE d$>51
 THEN RETURN
5020 GO SUB fuoco
5030 RETURN
```

Smontiamo il programma

Per la maggior parte i programmi di questo libro sono piuttosto brevi e abbastanza facili da seguire. Questo è un poco più serio (nel contenuto di programmazione)e vale la pena di dedicarvi un po' di studio, perché la tecnica usata per scriverlo è comune e potente, variamente nota come *top-down analysis* o *step-wise refinement (raffinamento dall'alto al basso)*..

L'idea è di cominciare a scomporre il programma in una sequenza di passi principali. Esaminiamo quindi ciascun passo e decidiamo se è possibile codificarlo direttamente in modo agevole oppure, in alternativa, se è utile scomporlo in pezzi più piccoli. Continuiamo il procedimento finché tutti i passi sono abbastanza piccoli da poter essere codificati senza difficoltà. Ogni passo ad ogni livello diventa un sottoprogramma.

In questo programma, il primo stadio di scomposizione ha un aspetto simile a questo:

```
                Inizializza le variabili
                Disegna le città e i sili
Finchè ci       ___        ___ Genera nuovi missili
sono ancora        |      |
anti-missili       |------|
e città         ___|      |___ Muovi i missili esistenti
                Stampa il punteggio
```

Scorrendo il listato, si vede che le inizializzazioni sono effettuate tra le linee 1 e 80. Potreste poi aspettarvi di trovare qualcosa come

```
100   GO SUB 500
```

per entrare nella routine di disegno, ma in realtà vedrete che è scritto

```
100   GO SUB disegno
```

e che "disegno" è una variabile posta a 500 in fase di inizializzazione. Questo rende il programma più leggibile ed è particolarmente pratico durante il debug. Per esempio, una volta fatto girare il programma [1], potete scrivere

```
LIST disegno
```

invece di dover ricordare

```
LIST 500
```

quando dovete esaminare quella particolare routine.

Le linee tra 110 e 160 formano semplicemente il ciclo a condizione iniziale "finché" nella descrizione scheletrica del programma. Co-

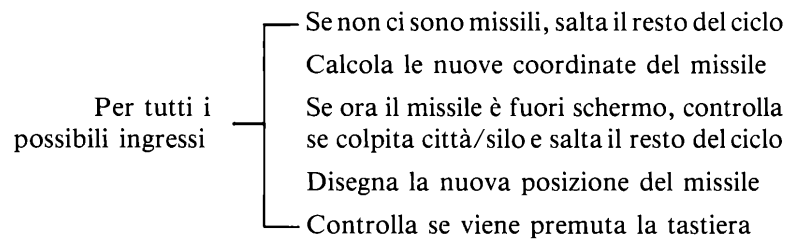
---

[1] NdT. In fase di debug il programma non girerà, se contiene degli errori, ma è sufficiente che arrivi ad eseguire l'area d'inizializzazione.

me potete vedere, nel ciclo ci sono due chiamate a sottoprogramma, una per generare nuovi missili (genmis) e l'altra, come previsto, per muovere i missili esistenti (movmis).

Guardiamo movmis, per esempio. Per muovere un missile dobbiamo sapere dove si trova. In effetti in un dato istante ci possono essere solo 20 missili e di ognuno sono conservate tre informazioni (la sua coordinata x, la coordinata y e l'angolo di attacco) in una matrice detta m. Per convenzione se la casella dell'angolo di attacco contiene il valore zero, allora là non c'è alcun missile. In questo modo dobbiamo controllare solo uno dei tre parametri per vedere se c'è un missile; successivamente, quando vogliamo distruggerne uno, dobbiamo solo azzerare l'angolo di attacco, per indicarne la disattivazione.

Dunque la scomposizione all'interno di "movmis" ha questo aspetto:

Per tutti i possibili ingressi:
- Se non ci sono missili, salta il resto del ciclo
- Calcola le nuove coordinate del missile
- Se ora il missile è fuori schermo, controlla se colpita città/silo e salta il resto del ciclo
- Disegna la nuova posizione del missile
- Controlla se viene premuta la tastiera

Guardando il listato, vedrete che due blocchi sono scritti in forma di sottoprogrammi: il "controlla se colpita città/silo" (detto contcolp) e il "controlla se viene premuta la tastiera" (detto pretas). Quest'ultimo sottoprogramma è presente perché ci serve sapere se l'utente ha appena premuto un tasto per attivare un silo.

Guardate ora "pretas":

Se non è premuto alcun tasto, allora RETURN
Se il tasto premuto non è né 1 né 2, allora RETURN
Fuoco con l'anti-missile

e vedrete che "fuoco" è un altro sottoprogramma.

Vedete come questo tipo di approccio e analisi ci permette di preoccuparci solo di un problema alla volta?

Lascerò a voi il compito di finire di ricreare lo scheletro di "genmis", "fuoco", e "prnscr".

Un accenno alle REM

Sviluppando un programma, l'istruzione REM è un notevole aiuto alla memoria. Il computer ignora tutte le REM, eccetto che durante il LIST; così potete inserire qualche messaggio per voi stessi, per rammentarvi perché una particolare istruzione si trovi proprio lì. Effettivamente non ho utilizzato molte REM nel corso di questo libro; ma la ragione è che tutti i programmi sono in ogni caso accompagnati da una discussione particolareggiata. Ma quando state scrivendo un programma per conto vostro, le REM sono una benedizione. Po-

tete utilizzare dei trucchi per far risaltare le REM all'interno di un listato: mettere una riga di asterischi REM *******, o usare i caratteri di controllo per averle stampate in un altro colore.

Modifiche e miglioramenti

1. Per il momento, quando attivate un silo, il programma non conferma di quale silo si tratti. Fate si che il silo attivato lampeggi (e che si disattivi pochi secondi dopo il fuoco).
2. Le tracce dei missili e degli anti-missili sembrano identiche. Date loro colori differenti (NB: le particolarità del sistema dei colori faranno sì che il tracciamento di una nuova scia possa cambiare il colore di parti di quelle precedenti. Non c'è modo di aggirare questo ostacolo, ma non ha grande importanza).
3. In versioni più sofisticate di questo gioco, i missili in arrivo sono MIRV, e si separano in testate multiple durante la discesa. Questo effetto non è difficile da ottenere, ma richiede un po' di attenzione.
4. La distruzione delle città è poco appariscente (vengono semplicemente spazzate via!). Che ve ne pare di fare comparire una nube a forma di fungo e poi qualche rovina?
5. Pensate a degli effetti sonori. (Ma questi rallenteranno molto il gioco).
6. Dato che ''pretas'' viene richiamata piuttosto raramente, può accadere che dobbiate tenere premuti i tasti 1 e 2 per qualche istante, prima di avere una risposta. Provate altre posizioni del programma ove mettere ''GO SUB pretas'' per migliorare il tempo di risposta senza rallentare troppo il resto delle operazioni.

23. PEEK e POKE

L'organizzazione interna dello Spectrum non è qualcosa di cui dobbiate preoccuparvi. Ma, se volete, potete far sì che la macchina vi dica quello che sta facendo e modificarlo per assecondare i vostri scopi

Potrei probabilmente scrivere un altro libro delle dimensioni di questo su PEEK e POKE. Ma la maggior parte sarebbe di gran lunga troppo tecnica per risultare utile. Sembra però un peccato evitare del tutto di parlare di queste caratteristiche estremamente importanti. Esplorando PEEK e POKE potete davvero imparare moltissimo su *come* funzionano i computer. Così, ovviamente, quello che vi serve è un'idea di come partire: come convincere lo Spectrum a rivelare alcuni dei suoi segreti più riposti.

Nel Capitolo 1 ho menzionato il fatto che i calcolatori immagazzinano le informazioni sotto forma di sequenze di 0 e 1. Ognuno di questi 0 e 1 è detto bit (contrazione di *b*inary dig*it* = cifra binaria). Potete pensare una stringa di 0 e 1 come un numero nel *sistema binario*, che è proprio come il sistema decimale, con l'eccezione che al posto di unità, decine, centinaia, migliaia, ecc., si usano unità, coppie, quartetti, ottetti e così via. Tra i Programmi preconfezionati troverete un programma sulla conversione binario/decimale che spiega qualcosa di più sull'argomento.

Non è necessario conoscere il codice binario, ma è indispensabile sapere che i computer lavorano con i *bit*. In effetti, generalmente operano su sequenze di bit, dette byte. Un byte è una sequenza di otto bit. Così 10110001 e 00111011 sono dei tipici byte.

Esistono 256 possibili byte differenti; se li convertite in decimale, ottenete i numeri tra 0 e 255. In effetti, ogni carattere viene rappresentato da esattamente un byte.

Un programma è proprio una sequenza di caratteri; così la macchina può memorizzarlo come sequenza di byte. Per assicurarsi che tutto rimanga nel giusto ordine, assegna a ogni byte un numero di riferimento, detto *indirizzo*. Così potreste immaginare che il programma, all'interno della macchina, assuma una forma analoga alla seguente.

Indirizzo	Byte
1	11001100
2	01110000
3	00000000
4	11101111

.....

Sfortunatamente l'"architettura" della macchina rende questo schema un poco troppo semplicistico.

Lo Spectrum consiste di un insieme di microchip, noti come:

ROM	(Read Only Memory = Memoria di sola lettura)
RAM	(Random Access Memory = Memoria ad accesso casuale)
CPU	(Central Processing Unit = Unità centrale di elaborazione)
SCL	(Sinclair Logic Chip = Circuito logico Sinclair)

La ROM contiene l'interprete BASIC; la RAM immagazzina il vostro programma e qualsiasi altra cosa sia da calcolare durante l'esecuzione; la CPU esegue la parte logica e aritmetica; l'SCL presiede all'interazione dei precedenti elementi.

La CPU e l'SCL non ci interessano, ma siamo effettivamente interessati alla ROM e alla RAM. Possiamo scoprire esattamente quali byte siano memorizzati in dati indirizzi della RAM e della ROM usando PEEK.

Il comando

```
PRINT PEEK 837
```

per esempio, stamperà il byte memorizzato all'indirizzo 837 (è 40, cioè 00101000 in binario). Stampa il corrispondente numero decimale, in realtà; quindi non c'è alcun male nel considerare un byte semplicemente un numero compreso tra 0 e 255.

Dove dovremmo guardare, per delle PEEK utili?

Gli indirizzi della ROM vanno da 0 a 16383. Esistono certamente dei buoni motivi per eseguire una PEEK nella ROM — potete scoprire come lo Spectrum dice alla TV di stampare un certo carattere e potete giocare con questo (ad esempio potete stamparlo in dimensioni quadruple rispetto al normale).

Ma gli indirizzi RAM sono più interessanti. Cominciano da 16384 e tutti quelli fino a 23754 (o più se è presente il microdrive) sono assegnati e utilizzati dalla macchina. Il vostro programma è posto in un indirizzo memorizzato nella variabile di sistema PROG, che si trova agli indirizzi 23635 e 23636. L'indirizzo d'inizio del programma infatti è:

```
PEEK 23635 + 256 * PEEK 23636
```

Senza microdrive, dovrebbe valere 23755. Controllate. Per scoprire che cosa è memorizzato nella RAM, fate fare il lavoro allo Spectrum. (Fatelo sempre, se potete!). Ecco un programma adatto allo scopo.

```
1000  LET q = PEEK 23635 + 256 * PEEK 23636
1010  LET r = PEEK q
1020  IF r > 23 THEN PRINT q; "□□"; r; TAB 12; CHR$ r
1030  IF r < = 23 THEN PRINT q; "□□"; r; TAB 12; "?"
1040  LET q = q + 1
1050  GO TO 1010
```

Ho usato numeri di linea elevati perché l'idea è quella di scrivere un programma di prova prima di questo, e di vedere come e dove sia stato memorizzato usando GO TO 1000.

Battete il programma precedente, e un programma di prova:

```
10  REM inizio
20  PRINT AT 0, 10; "*"
```

Battete ora GO TO 1000 e guardate. Otterrete uno stampato come questo:

```
23755  0    ?
23756  10   ?
23757  8    ?
23758  0    ?
23759  234  REM
23760  105  i
23761  110  n
23762  105  i
23763  122  z
23764  105  i
23765  111  o
23766  13   ?
23767  0    ?
23768  20   ?
23769  23   ?
23770  0    ?
23771  245  PRINT
23772  172  AT
23773  48   0
23774  14   ?
23775  0    ?
23776  0    ?
```

A questo punto sta domandando "scroll?", ma diamo uno sguardo a quello che c'è sullo schermo.

Possiamo vedere il nostro programma — o almeno la maggior parte — che si ricostruisce nella terza colonna: REM, i, n, i, z, i, o,..., PRINT, AT, 0,... Ma ci sono anche altre cose. La terza colonna non sembra aiutarci a capirle; ma la seconda sì. Per esempio all'indirizzo 23756 troviamo 10 e a 23768 troviamo 20: presumibilmente questi sono i numeri di linea (la questione si presenta un po' più complicata con numeri di linea più alti); mentre l'8 in 23757 e il 23 in 23769 sono in realtà il numero di caratteri delle linee nella rappresentazione interna del calcolatore: vedi *Manuale*, Capitolo 33.

I punti interrogativi, nella terza colonna, si trovano là per alcuni buoni motivi: se togliete "IF r > 23" nella linea 1020 ed eliminate la 1030, il programma non funziona. Questo perché si tratta di caratteri di *controllo* (p. es. predisposizione dei colori) e la macchina tenta di eseguirli come istruzione, ottenendo così spesso delle cose prive di senso e fermandosi con un messaggio d'errore. Comunque potete consultare il codice dei caratteri sul *Manuale*, pagg. 239-245. Per esempio all'indirizzo 23766 abbiamo il carattere 13 che significa ENTER — sì, lo abbiamo effettivamente premuto vero?

Dunque, ci sono dei dettagli da risolvere, ma possiamo vedere dove il programma è memorizzato, ed è veramente *inserito* là. Per vedere di più, premete "y" per far scorrere lo schermo (una caratteristica che rende estremamente utile il nostro programma). Ora abbiamo:

```
23777  0   ?
23778  0   ?
23779  0   ?
23780  44  ,
23781  49  1
23782  48  0
23783  14  ?
23784  0   ?
23785  0   ?
23786  10  ?
23787  0   ?
23788  0   ?
23789  59  ;
23790  34  "
23791  42  *
23792  34  "
23793  13  ?
```

Potete continuare a listare per parecchio tempo. Presto la routine alla linea 1000 inizierà a listare se stessa...

Provate qualche altro programma nelle linee 10, 20, 30, ecc. Lasciate le linee 1000-1040 così come sono e controllate i vostri programmi usando GO TO 1000.

Una cosa veramente strana è il modo in cui i *numeri* vengono memorizzati. Il 10 di PRINT AT 0, 10 sembra occupare gli indirizzi 23781-23788. Parrebbe uno spazio eccessivo per un numero così piccolo; ma è collegato al fatto che lo Spectrum può maneggiare numeri in *virgola mobile* (decimali come 23.567) e quindi usa un codice adatto. Potete divertirvi per ore, solo cercando di scoprire come un dato numero sia effettivamente memorizzato.

Ma la cosa più bella di tutto ciò è che, ora che sapete *dove* si trova un dato byte di programma, potete *modificarlo*. L'istruzione che vi dà questo terribile potere sul povero Spectrum è:

POKE

Vediamola in azione

Che cosa fa il nostro precedente programmino? Stampa * nella posizione 0, 10. Premete RUN e vedrete!

Operiamo con una POKE. Aggiungiamo due linee:

```
30  POKE 23791, 96
40  GO TO 10
```

Fatelo girare battendo:

GO TO 30

Invece di * viene stampato £.
Perché? Perché la linea 30 impone all'indirizzo 23791 (dove era memorizzato *) il nuovo carattere di codice 96, che è £.
Se chiedete alla macchina LIST, trovate che la linea 20 del programma ora è:

20 PRINT AT 0, 10; "£"

Sperimentiamo ancora un poco. BREAK; sbarazzatevi delle linee 30 e 40. Riportate la linea 20 alla sua forma originaria usando EDIT. Contemporaneamente, trasformate lo 0 in 1 per avere:

20 PRINT AT 1, 10; "*"

Ora eseguite i PEEK con GO TO 1000. Fate scorrere.
Noterete il cambiamento. Gli indirizzi 23773 e 23777 sono diventati

23773	49	1
23777	1	?

Tutto il resto è come prima.
Per verificare quanto avete appena imparato, aggiungete le seguenti linee di programma:

```
30 POKE 23773, 49
40 POKE 23777, 1
50 GO TO 20
```

Ritrasformate la linea 20 in PRINT AT 0, 10; "*". Ora premete RUN.
Lo schermo dovrebbe mostrare *due* asterischi, alle linee 0 e 1. Si esegue il primo programma, stampando alla linea 0; quindi si impongono con la POKE i valori necessari alla linea 1; poi GO TO 20 fa stampare nella nuova linea.
Provate con LIST. Ancora una volta trovate che la linea 20 è diventata:

20 PRINT AT 1, 10; "*".

Naturalmente non potete eseguire con successo una POKE sulla ROM — "Sola lettura" significa proprio ciò che dice! Ed è un bene, altrimenti una POKE accidentale potrebbe rovinare l'interprete BASIC. Di conseguenza potete eseguire POKE senza paura, se vi prende il capriccio, per vedere quello che succede.
Le possibilità aperte da POKE sono vastissime. Ma per maneggiarle nel modo più appropriato dovreste sapere qualcosa di più sui dettagliati codici interni dello Spectrum. I capitoli 33 e 34 del *Manuale*

contengono le informazioni fondamentali che vi servono. Se siete abili abbastanza, potete estrapolare il resto usando la precedente routine PEEK, linee 1000-1050. Non ho intenzione di dire altro, poiché è meglio che ci arriviate da soli, col vostro lavoro, piuttosto che attraverso complesse descrizioni fatte da qualcun altro. Tuttavia, come in altre parti di questo libro, ho voluto semplicemente incanalare il vostro lavoro lungo direzioni valide. Per il momento, avete visto che PEEK e POKE non sono quel mistero spaventoso che spesso sono fatte sembrare. È solo che scavano più in profondità nel *sistema* con il quale funziona lo Spectrum. Ora, fate voi stessi prove ed esperimenti: PEEK e POKE rappresentano una sfida affascinante nonché potenzialmente gratificante.

24. Suggerimenti

Un paio di problemi tipici e qualche trucco utile

1. È naturale cercare di elevare al quadrato un numero x calcolando x↑2. Questo va bene per x positivi ma, benché il quadrato di un numero negativo abbia perfettamente senso, in questo caso la freccia non funzionerà. (Ciò avviene a causa del fatto che x↑a viene calcolato come EXP (a * LN x) e il logaritmo di un numero negativo non è definito). Usate x * x invece di x↑2 se c'è la possibilità che x sia negativo.

2. Nello stesso ordine d'idee, se x è un intero, x↑2 ha la pessima abitudine di non essere per niente esatto, e potete finire nel tipo di problemi discussi in Debugging V. Analogamente x↑3: *può* davvero essere meglio scrivere x * x * x.

3. Se usate una REM in una linea multi-istruzione, ricordatevi che *tutto* quello che segue (su quella stessa linea) viene ignorato dal computer. Così

```
10  REM inizio: LET x = 99
```

non ha alcun effetto. Mentre

```
10  LET x = 99: REM inizio
```

pone x a 99.

4. Qualsiasi istruzione IF/THEN funziona così: si esegue *l'intera* linea dopo il THEN, posto che sia soddisfatta la condizione IF, ma *niente* di quello che segue THEN viene eseguito se la condizione non è vera. Così la linea

```
10  IF x = 0 THEN LET y = 0: IF x < > 0 THEN LET y = 1
```

pone y a 0 se x è 0, ma viene *completamente ignorata* se x < > 0.

Per contro, con le linee separate

10 IF x = 0 THEN LET y = 0

20 IF x < > 0 THEN LET y = 1

si porrà y a 1 per ogni x diverso da 0. Se le vostre istruzioni condizionali fanno le bizze, controllate per prime le linee multiple.

5. In un'istruzione IF/THEN, la macchina calcola l'*intera* espressione tra IF e THEN prima di provare a stabilire se è vera. Ciò può portare a imprevisti messaggi d'errore. Per esempio

10 IF x$ < > " " AND VAL x$ = 5 THEN STOP

crolla se x$ è vuota — anche quando x$ < > " " è falsa, e dunque rende falsa l'intera condizione qualsiasi sia VAL x$, la macchina insiste nel calcolare VAL x$. Facilmente potreste *credere* di aver protetto il programma contro condizioni catastrofiche, ma potreste esservi sbagliati...

6. Ci sono due modi per ottenere uno spazio bianco. Uno è il tasto SPACE, l'altro è il carattere grafico 8 con CAPS SHIFT. *Questi non sono lo stesso carattere*, per quanto riguarda lo Spectrum. Dunque la ricerca di spazi con un'istruzione

10 IF n$ (i) = "□" THEN GO TO 5000

andrà a vuoto se lo spazio nella posizione n$ (i) non è dello stesso tipo dello spazio alla linea 10 del vostro programma. Sfortunatamente non potete individuare l'errore su un listato!

7. Potete memorizzare su nastro una figura con

SAVE "Figura" SCREEN$

È terribilmente facile tentare di ricaricarla con

LOAD "Figura"

Nel qual caso appare sì il messaggio "Bytes: Figura", ma in pratica non succede niente. Naturalmente quello che avreste dovuto fare è

LOAD "Figura" SCREEN$

8. La qualità della visione a colori è molto sensibile alla qualità del vostro televisore e alla sintonia. Su dei televisori a colori portatili molto economici, potreste scoprire che la figura non è assolutamente soddisfacente; *non* comprate quindi un piccolo TV a colori per usarlo specialmente come monitor, prima di aver controllato che va bene. Se in un negozio non vi permettono di collegare il vostro Spectrum prima dell'acquisto, rivolgetevi a un altro negozio!

9. Se avete in memoria un programma e volete memorizzarlo su nastro con SAVE, ma avete dimenticato dove si trova posizionato il nastro — e temete di lasciare uno spazio troppo lungo o di cancellare qualcos'altro d'importante — battete

VERIFY "Cancellazione"

dove "Cancellazione" deve essere un nome non usato per alcun programma. Fate partire il nastro, e guardate i messaggi per vedere dove vi trovate. Quando raggiungete il punto desiderato, fermate il nastro; premete BREAK; poi caricate (LOAD) il vostro programma come al solito.

Trucchi con le variabili di sistema

1. Potete ottenere un "beep" di tastiera molto più soddisfacente se date il comando diretto:

POKE 23609, 50

Numeri diversi da 50 danno effetti un poco diversi, ma 50 va abbastanza bene.

2. Potete accelerare l'auto-ripetizione con il comando diretto

POKE 23562, 2

(per esempio). Trasformate il 2 in 1 o 3 per un'auto-ripetizione più rapida o più lenta; portatelo a 0 per eliminarla completamente.

3. Potete ridurre l'intervallo di tempo che la macchina attende prima dell'auto-ripetizione battendo (per esempio):

POKE 23561, 20

Per ridurre il ritardo, diminuite il valore 20, aumentatelo in caso contrario.

4. Alcuni programmi chiedono uno SCROLL automatico. (Molti dei giochi dello ZX81 *usano* lo scroll e potreste volerli trasferire allo Spectrum) Ma non c'è un comando SCROLL.

Potete comunque convincere la macchina a fare scorrere il programma con la routine:

```
1000  PRINT AT 21, 0
1010  POKE 23692, 2
1020  PRINT
```

In effetti, imponete POKE 23692 con un valore maggiore di 1, nel momento in cui si riempie lo schermo, e provate poi a stampare una nuova linea: si produce lo scroll senza ingresso da tastiera.

5. Il comando DRAW x, y, ottimo così come è, traccia una linea dalla posizione corrente x0, y0 alla nuova posizione x0 + x, y0 + y. Cioè, x e y sono gli *incrementi* necessari, non le nuove coordinate; un metodo per andare dalla posizione corrente a una nuova posizione x, y consiste nello scrivere:

DRAW x — PEEK 23677, y — PEEK 23678

Il vantaggio di questo sistema è che funziona anche quando avete perso traccia di dove fosse l'ultima posizione PLOT (cosa abbastanza facile) e che non va soggetto a errori di arrotondamento che si sommano se lo usate in un ciclo, ad esempio per disegnare delle curve. Naturalmente gli indirizzi 23677 e 23678 contengono le coordinate dell'ultimo punto disegnato!

25. Quello che vi ho tenuto nascosto

E ora si rivela la terribile verità

C'è molto, molto di più nel vostro Spectrum di quanto non ho potuto anche solo accennare in questo libro. Avrei volentieri aggiunto altro, ma ciò avrebbe reso l'opera della lunghezza di 17 volumi, e ciascuno avrebbe avuto un costo dieci volte superiore... Comunque, dal momento che siete sopravvissuti a *questo* testo, il *Manuale* Sinclair avrà molto più senso. (Non sto biasimando il *Manuale* — ma, per definizione, un manuale deve dirvi *tutto* e questo comporta descrizioni piuttosto condensate).

Per esempio, non ho detto niente delle funzioni matematiche come EXP, COS, TAN, LN; non ho detto niente delle funzioni-utente DEF FN a (x, y, z...) che vale la pena di conoscere; non ho detto quasi niente degli attributi ATTR; ho citato solo uno degli utilizzi di USR (ma gli altri portano al linguaggio macchina); ho usato qualche matrice multidimensionale, ma non l'ho mai spiegata; non ho detto niente su INVERSE; e non ho spiegato LOAD e SAVE, dal momento che il *Manuale* le tratta molto chiaramente e che, in ogni caso, il sistema sopporta anche le operazioni più sconsiderate.

Ma la cosa più importante che voglio qui puntualizzare è che se, in un listato, vedete un comando che non capite, *potete ugualmente copiare il listato e farlo girare*. Se avete spirito d'avventura, potete modificare il comando che vi lascia perplessi, e vedere l'effetto che fa: in questo modo potrete persino scoprire il suo funzionamento. Siate coraggiosi.

Programmi preconfezionati

I programmi che seguono tendono all'illustrazione di varie caratteristiche dello Spectrum e a mostrarvi il tipo di cose che si possono ottenere. Ognuno è completo in se stesso e necessita solo di essere copiato e mandato in esecuzione. Non è necessario comprendere subito tutte le istruzioni del listato.

Tuttavia, una volta terminato questo libro, dovreste essere in grado di analizzare come funzionano questi programmi. Non sarà sempre facile, comunque: è spesso difficoltoso abituarsi allo stile di programmazione di un'altra persona. Un buono stile richiede chiarezza, in parte perché è un compito scoraggiante tentare di modificare un programma che sembra il *Finnegan's Wake* scritto in cinese, e in parte perché si tende a commettere meno errori.

Ho deliberatamente lasciato numerosi programmi con numeri di linea "disordinati". È incredibile quanto spesso s'intrufolino errori quando rinumerate un programma. Inoltre volevo sottolineare che *non* è necessario che usiate numeri di linea multipli di 10. Le sole ragioni per farlo, oltre all'abitudine o al manierismo, sono di lasciare spazio per modifiche da apportare al programma, o per inserire indicatori durante il debugging.

I listati sono stati riprodotti direttamente da una stampante collegata allo Spectrum, e appaiono esattamente come sul video.

Una parola sul *colore*. Prima di far girare un qualsiasi programma, potete scegliere i colori di cornice, sfondo e caratteri con una istruzione diretta dalla tastiera — per esempio:

BORDER 3: PAPER 5: INK 1

Non ho inserito questo tipo di comandi nei programmi: se volete potete farlo voi. Lo scopo è quello di risparmiarvi tempo nel battere il programma, e di renderlo più chiaro — anche se il display non è proprio così spettacolare. Analogamente, alcuni dei programmi sono piuttosto semplici: dovreste cercare di scoprire *che cosa fanno*, e non solo copiarli pedissequamente.

I programmi non sono presentati in un ordine particolare.

Buon lavoro!

Area di un triangolo

Siano a, b, c i lati di un triangolo; l'area si può calcolare con la formula $\sqrt{s(s-a)(s-b)(s-c)}$ ove $s = 1/2(a + b + c)$. Questo programma usa la formula precedente per calcolare l'area di un triangolo i cui lati vengono inseriti in sequenza.

```
 10 INPUT a
 20 PRINT "a= ";a
 30 INPUT b
 40 PRINT "b= ";b
 50 INPUT c
 60 PRINT "c= ";c
 70 LET s=.5*(a+b+c)
 80 LET x=s*(s-a)*(s-b)*(s-c)
 90 IF x<0 THEN GO TO 110
 95 PRINT "L' area e' ";SQR x
100 STOP
110 PRINT "E' impossibile costr
uire un triangolo con questi lat
i"
```

Note al programma

1. I programmi che applicano una formula e ne stampano il risultato sono così trasparenti che sarebbe una truffa considerarli "veri" programmi! Sono più simili a routine automatizzate per calcolatrici tascabili. Ma, in questo stadio, ho pensato che un programma ovvio non avrebbe fatto nessun danno.
2. Potete agevolmente modificare questo tipo di programma per calcolare qualsiasi formula ragionevole. Forse questo è un modo di rendere le formule matematiche un poco più interessanti e di ottenere dei contenuti educativi dallo Spectrum. Ecco alcuni suggerimenti per dei possibili programmi:
   - (a) L'area della superficie di una sfera di raggio r è $4 \pi r^2$ [π è PI sullo schermo: tasto M in modo esteso].
   - (b) Il volume di una sfera di raggio r è $4/3 \pi r^3$.
   - (c) Il volume di un cilindro di raggio r e altezza h è $\pi r^2 h$.
   - (d) Il volume di un cono di raggio r e altezza h è $1/3 \pi r^2 h$.
   - (e) Le soluzioni dell'equazione di secondo grado $ax^2 + bx + c = 0$ sono date da:

     $$x = (-b \pm \sqrt{(b^2 - 4ac)}) / 2a$$

     (Calcolate separatamente la radice + e quella —. Se $b^2 - 4ac < 0$, le radici sono complesse, e dovrete poter affrontare il caso, o stampando "Radici complesse", oppure inventando un pezzo supplementare di programma per calcolarle, se conoscete i numeri complessi. Dovrete anche poter trattare la possibilità che $a = 0$).
   - (f) La somma $1 + 4 + 9 + \ldots + n^2$ è uguale a $1/6\, n (n + 1)(2n + 1)$. Inserite n e stampate il risultato. [Per confronto, calcolate il risultato anche sommando la serie con cicli FOR/NEXT, dopo aver letto il Capitolo 6].
   - (g) Il periodo di oscillazione di un pendolo di lunghezza p è $T = 2\pi \sqrt{p/g}$ dove g rappresenta l'accelerazione di gravità: 981 cm/sec/sec
   - (h) In effetti g varia da luogo a luogo: un'approssimazione più

accurata è che dipende dalla latitudine L e dall'altezza h sul livello del mare, secondo la formula:

$$g = 980.616 - 2.5928\cos(2L) + 0.0069\cos^2(2L) - 0.0003h$$
cm/sec/sec.

Scrivete un programma per calcolare g e T, una volta che si conoscano L, h e p.

Il conto in banca

Se i triangoli non sono il vostro genere, perché non scrivere un programma per calcolare il vostro conto in banca?

```
  10 PRINT "Il bilancio preceden
te e' ";
  20 INPUT b
  30 PRINT b
  40 PRINT "Elenco debiti "
  50 INPUT d
  60 IF d<0 THEN GO TO 90
  70 LET b=b-d
  80 GO TO 50
  90 PRINT "Elenco crediti "
 100 INPUT d
 110 IF d<0 THEN GO TO 140
 120 LET b=b+d
 130 GO TO 100
 140 PRINT "Il bilancio corrente
 e' ";b
```

Note al programma

1. Le linee 60 e 110 sono dei *terminatori*. Se inserite un numero negativo, la macchina capisce che avete finito l'enumerazione. Questo numero negativo *non* viene inserito nel bilancio!
2. I numeri del dare sono principalmente quelli delle matrici del libretto degli assegni. Non dimenticate comunque gli ordini fissi. (Un primo ovvio miglioramento è quello di introdurli nel programma. Provate a vedere se riuscite a cavarvela). E i prelievi dallo sportello automatico.
3. Avere: non dimenticate il vostro stipendio! Magari riuscite a inserire anche questo.
4. I vostri bambini più piccoli possono giocare alla banca per ore usando questo programma.
5. Aggiungendo altre due linee

```
 55  PRINT d
105  PRINT d
```

   la macchina stamperà la lista delle voci. Se avete più di 20 voci lo schermo si riempie, ed è necessario fare lo scrolling per continuare.

Caccia alla tigre

Nelle oscure profondità della foresta "spectrale" si nasconde una terribile tigre. Riuscite a trovarla?

```
  10 PAPER 7: INK 8: BORDER 1
  20 INPUT "scegliete le dimensi
oni della foresta: max 16 ";d
  25 IF d>16 THEN LET d=16
  30 FOR i=0 TO 8*d STEP 8
  35 PLOT i+64,32
  40 DRAW 0,8*d
  50 PLOT 64,i+32
  60 DRAW 8*d,0
  70 NEXT i
  80 LET z$="0123456789111111"
  90 LET y$="          0123456":
 REM 10 spazi
 100 PRINT AT 19,8;z$( TO d)+" "
+"x"
 110 PRINT AT 20,8;y$( TO d)
 120 PRINT AT 16-d,6;"y";AT 17-d
,0
 130 FOR i=1 TO d
 140 PRINT TAB 6-(d-i>9);d-i
 150 NEXT i
 200 LET mx=INT (d*RND)
 210 LET my=INT (d*RND)
 220 INPUT "Dov'e la tigre? ";x;
" ";y
 225 IF x>d OR y>d THEN GO TO 22
0
 230 LET dd=((mx-x)*(mx-x)+(my-y
)*(my-y))/d/d
 240 LET dd=dd*8: IF dd>4 THEN L
ET dd=4
 250 PRINT PAPER dd+2;AT 17-y,8+
x; OVER 1;" "
 255 IF mx=x AND my=y THEN GO TO
 300
 260 GO TO 220
 300 PRINT AT 17-my,8+mx; OVER 1
;"*"
 310 PRINT AT 0,0; FLASH 1;"Esat
to!!!!"
 320 FOR i=1 TO 30
 330 BEEP .1/i,i
 340 NEXT i
```

Note al programma

1. Potete scegliere qualsiasi dimensione della foresta da 1 a 16. Se inserite un numero maggiore, vi viene assegnata una dimensione di 16. La foresta viene disegnata con le coordinate x e y lungo la parte inferiore e laterale dello schermo (usando le linee 30-70 per il bosco e 80-150 per le coordinate). Notate come la costruzione della figura sia la parte preponderante del programma!
2. Quando vi viene domandato "Dov'è la tigre?" dovete inserire due numeri compresi tra 0 e d — 1. Se inserite dei numeri al di fuori di questo intervallo, vengono ignorati.
3. La macchina a questo punto stampa un quadratino colorato nella posizione indicata. Il colore dà un indizio sulla distanza: il rosso significa che siete molto vicini, il magenta abbastanza vicini, il verde abbastanza lontani, e il giallo molto lontani.
4. Continuate finché non localizzate la tigre. A questo punto avete un "*", un messaggio di congratulazione lampeggiante e un commento musicale per sottolineare la cattura. (La musica si

trova alle linee 320-340: potete usarla in altri programmi per ottenere gli stessi suoni).

5. Per tentare ancora una volta, RUN.

Progetti

1. Quando siete veramente vicini alla tigre, fate in modo che il quadratino rosso lampeggi. Potete farlo alla linea 250. Inserite dopo OVER 1:

   FLASH (d < qualcosa)

   e fate degli esperimenti per trovare che valore deve avere ‘‘qualcosa’’ per un effetto piacevole.
2. Trasformate il messaggio in qualcosa di violento.
3. Modificate la musica: costruite una selezione di motivi tra i quali la macchina sceglie a caso, per non annoiarsi.

Compositore

Questo programma permetterà allo Spectrum di suonare della musica. Le note vengono inserite una alla volta, come singole lettere o lettere seguite da# per indicare i diesis. Premete ** per chiudere la sequenza delle note e fare iniziare l'esecuzione

```
  10 DIM s(7)
  11 DIM n$(2)
  12 DIM n(1500)
  20 LET s(1)=-3: LET s(2)=-1: L
ET s(3)=0: LET s(4)=2
  30 LET s(5)=4: LET s(6)=5: LET
 s(7)=7
  35 FOR q=1 TO 1500
  40 INPUT "inserite una nota";n
$
  42 IF n$="**" THEN GO TO 200
  45 LET i=0
  50 LET p=CODE n$-64
  55 IF p>10 THEN LET p=p-32
  60 IF n$(2)="#" THEN LET i=1
 100 LET n(q)=s(p)+i
 110 NEXT q
 200 FOR r=1 TO q
 210 BEEP 0.5,n(r)
 220 NEXT r
```

I particolari sul funzionamento sono spiegati nel capitolo dedicato alle matrici.

Miglioramenti e modifiche

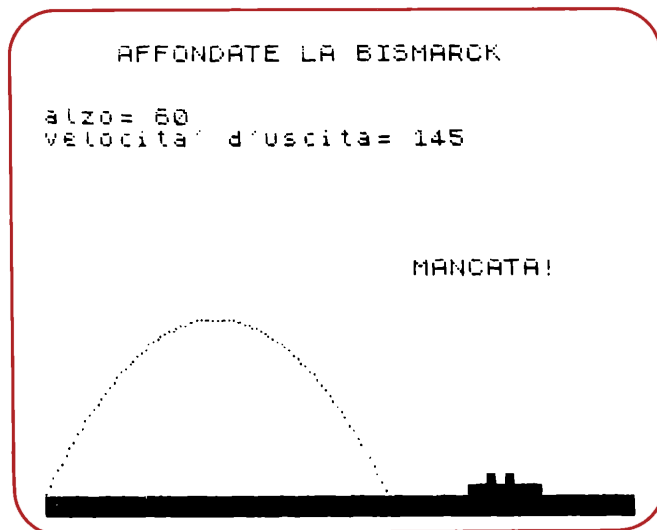
1. Sostituite la linea 40 con:

   40 LET n$ = CHR$ (INT (RND * 7) + 65)

   Ora il programma compone il suo pezzo musicale, contenente esattamente 1500 note. Nel risultato non si hanno diesis, così suona piuttosto piatto.
2. La lunghezza di ogni nota è fissata a 0.5 secondi. Riuscite a costruire una routine che permetta all'utente di inserire la lunghezza desiderata di ogni nota, insieme alla nota stessa? Vi servirà una seconda matrice, della stessa lunghezza din, ove porre i valori relativi alla durata. (Nel modello da 16K, le due matrici da 1500 elementi usciranno dalla memoria. Fate ogni matrice di lunghezza 750).
3. Usate una tecnica simile a quella di (1) per produrre musica casuale con durata casuale delle note.
4. Inserire la musica una nota per volta, può diventare un po' noioso. Riuscite a escogitare una tecnica in cui si può inserire la musica sotto forma di singola sequenza?
5. Se le note potessero essere modificate singolarmente in modo agevole, questo potrebbe diventare un utile ausilio alla composizione. Sapreste scrivere un "modificatore di melodie" che vi permetta l'inserimento di, per esempio, 38, A# per indicare "trasforma la 38° nota in A#"?
6. Ampliate la limitante estensione a una sola ottava. Ancora una volta, vi serviranno più informazioni per nota. Per esempio A, 0 potrebbe significare "*la* nell'ottava del *do* centrale" e C#, 2 potrebbe significare "*do* diesis due ottave sopra il *do* centrale".

Affondate la Bismarck

Una pattuglia di stanza nel Mare del Nord... improvvisamente dalla nebbia appare il vascello nemico, a circa un miglio di distanza. Dite ai vostri cannonieri di stabilire l'alzo e la velocità del proiettile dei loro cannoni. Riuscirete ad affondare la *Bismarck*?

```
  10 PRINT TAB 4;"AFFONDATE LA B
ISMARCK"
  20 LET t=15*(1+RND)
  30 PRINT AT 21,0; INK 5; INVER
SE 1;"
       ": REM 32 spazi
  40 PRINT AT 20,t-2; INK 4;"▄██▄
▀"
  50 INPUT "alzo= ";al
  60 PRINT AT 3,0;"alzo= ";al
  70 INPUT "velocita' d'uscita=
";v
  80 PRINT "velocita' d'uscita=
";v
  90 LET a=v*COS (PI*al/180)
 100 LET b=v*SIN (PI*al/180)
 110 FOR j=0 TO b/4.9 STEP .3
 115 LET c=(b*j-4.9*j*j)
 120 IF a*j>2560 THEN GO TO 190
 130 IF c>1600 THEN GO TO 170
 140 INK 2
 150 PLOT .1*a*j,.1*c+8
 160 BEEP 0.005,c/100+10
 170 NEXT j
 180 IF ABS (a*b/4.9-t*80)<250 T
HEN GO TO 210
 190 PRINT AT 10,20;"MANCATA!"
 200 STOP
 210 FOR j=0 TO 15
 220 PRINT AT 20-j,t-2;"glug": B
EEP .3,6-3*j
 230 NEXT j
```

Note al programma

1. Inizia disegnando il mare, e la silhouette di una nave in una posizione casuale. La linea 20 sceglie la posizione, la 30 "stampa" il mare e la 40 la nave.

2. 50-80 riceve in ingresso e stampa i valori per l'alzo ''al'' e la velocità iniziale ''v''. Il giocatore, quando interrogato dalla macchina, deve inserire questi dati dalla tastiera: ''al'' è in gradi e deve essere compreso tra 0° a 90°; v è in metri al secondo e deve essere positivo.
3. 90-170 calcola e disegna la traiettoria dell'ogiva a intervalli di 0.3 secondi — vedi *Note matematiche*, di seguito.
4. 180 vi segnala ''colpito'' se il vostro proiettile arriva alla linea orizzontale entro 250 metri dal centro della nave. Modificate il valore 250, diminuendolo per un gioco più difficile o aumentandolo per uno più facile.
5. 190-230 sono le procedure d'uscita relative a ''COLPITO'' o ''MANCATO''.
6. Per iniziare o per continuare premete RUN seguito da ENTER, come al solito.
7. Lo schermo corrisponde a 2560 metri.
8. Il programma è abbastanza ''comprensivo'' verso l'utente, e continua a funzionare anche se il proiettile esce dalla parte superiore dello schermo (grazie alla linea 120).
9. I caratteri grafici alla linea 40 sono i caratteri 3 e 1 con CAPS SHIFT.
10. ''PI'' alle linee 90 e 100 è il tasto M in modo esteso, *non* i tasti P e I!

Note matematiche

Il cammino della testata viene calcolato usando una formula matematica precisa (in assenza di resistenza aerodinamica) per un corpo sottoposto all'accelerazione di gravità — uguale a 9.8 m/sec/sec. Alla linea 110, j rappresenta il tempo; a la componente orizzontale della velocità e b quella verticale; la formula PI * al / 180 converte da gradi a radianti. La linea 150 calcola l'altezza al tempo j e la converte nelle coordinate dello schermo. (Ogni pixel di PLOT è un quadrato di 10 metri). Alla linea 110, b / 4.9 è il tempo impiegato a raggiungere la linea orizzontale. Alla linea 180, a * b / 4.9 è la distanza percorsa in quell'istante; 80 * t serve per convertire t in coordinate pixel.

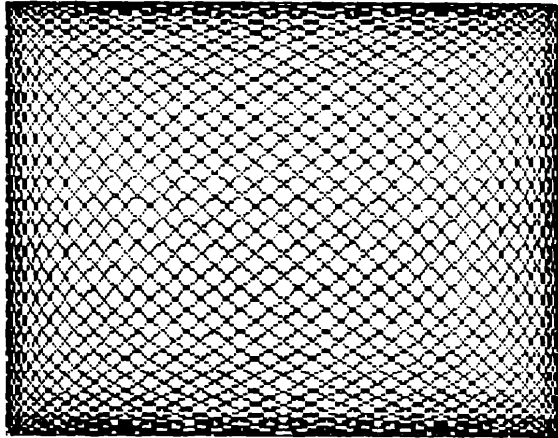
Se vi incuriosisce qualche altra caratteristica del programma, provate a modificarla per vedere il risultato.

Notate che, mentre *giocare* a un gioco come questo con il computer ha un valore educativo marginale (premere il tasto giusto, valutare angoli e distanze), *scriverlo* o *capirlo* richiede qualche conoscenza di programmazione e di matematica, il che lo rende un divertente ausilio didattico sia a scuola che in casa.

Figure di Lissajous

Ecco una modifica alla routine PLOT, che produce delle curve meravigliose.

Sono state inventate da Nathaniel Bowditch nel 1815 e presero nome da Jules-Antoine Lissajous che le reinventò nel 1857. Illustrano un altro modo di disegnare curve interessanti, usando quella che i matematici chiamano *parametrizzazione*. Ciò che significa è che si fanno dipendere le variabili x e y da una variabile t, per poi disegnare x, y per i diversi valori di t.

```
  10 INPUT "primo numero";p
  20 INPUT "secondo numero";q
  30 INPUT "sfasamento";r
  40 LET t=0
  50 LET m=p
  60 IF q<p THEN LET m=q
  70 LET x=127+120*COS (t*PI/180
*p+r)
  80 LET y=87+80*SIN (t*PI/180*q
)
  90 IF t=0 THEN PLOT x,y
 100 IF t>0 THEN DRAW x-PEEK 236
77,y-PEEK 23678
 110 LET t=t+10/m
 120 GO TO 70
```

Note al programma

1. Per ottenere dei risultati esteticamente validi, p e q devono essere numeri interi.
2. r può avere qualsiasi valore, ma tra 0 e 3 i risultati sono migliori.
3. Un buon punto di partenza è p = 5, q = 7 e r = 3.
4. "PI" è il tasto M in modo esteso.
5. Per fermarsi, BREAK + CAPS SHIFT
6. Per avere risultati più spettacolari (ma con tempi di esecuzione più lunghi) provate p = 31, q = 29, r = 0.
7. La linea 100 calcola lo spostamento tra il punto precedente e quello successivo, in modo da poterli congiungere con DRAW. Le variabili di sistema agli indirizzi 23677 e 23678, controllate da un PEEK, contengono le ultime coordinate PLOT. È questo un accorgimento estremamente utile.

Dadi del Monopoli

Lo Spectrum può fare statistica, grazie al generatore di numeri casuali RND. Questo programma simula il lancio di una coppia di dadi.

Se avete giocato a Monopoli®, vi ricorderete che per sapere di quanto deve muoversi il vostro pezzo, dovete lanciare due dadi e sommare il risultato. Potreste anche aver notato che il totale di 7 è più comune di qualsiasi altro. Infatti, su 36 lanci dei dadi, dovreste aspettarvi che il totale di

2 capiti in media 1 volta
3 capiti in media 2 volte
4 capiti in media 3 volte
5 capiti in media 4 volte
6 capiti in media 5 volte
7 capiti in media 6 volte
8 capiti in media 5 volte
9 capiti in media 4 volte
10 capiti in media 3 volte
11 capiti in media 2 volte
12 capiti in media 1 volta

Naturalmente, di solito non otterrete esattamente queste cifre, ma con molti lanci dovreste arrivarci vicino... in media!

Potete usare lo Spectrum per esplorare il mondo della statistica, *simulando* questo genere di cose tramite i numeri casuali. Questo programma "lancia" due dadi 144 volte (4 volte 36, per semplificarvi la vita); conta quante volte il totale vale 2, 3, 4, ecc.; disegna il risultato in un istogramma; e inoltre confronta il risultato effettivo con le previsioni teoriche.

```
  10 DIM a(11)
  20 FOR j=1 TO 144
  30 LET d=1+INT (6*RND)+INT (6*
RND)
  40 LET a(d)=a(d)+1
  50 NEXT j
  80 FOR j=2 TO 12
  90 LET q=a(j-1)
 100 LET q0=INT (q/2)
 110 LET q1=q-2*q0
 120 FOR t=1 TO q0
 130 PRINT AT 18-t,4+2*j;"▌"
 140 NEXT t
 150 IF q1=1 THEN PRINT AT 17-q0
,4+2*j;"▖"
 160 NEXT j
 170 PRINT AT 19,8;"2 3 4 5 6 7
8 9 1 1 1"
 180 PRINT AT 20,24;"0 1 2"
```

Nota al programma

L'istogramma indica il numero relativo di eventi dei totali 2, 3, 4, ..., 12. La forma teorica è triangolare con un picco al 7. Quanto effettivamente vi si avvicina? Otterrete la stessa forma, dopo un'altra esecuzione?

Test aritmetici

Ecco un modesto esempio di programma "educativo".

Questo programma sottopone all'operatore (vostro figlio di sette anni) un'addizione che comporta due numeri casuali di due cifre; controlla se la risposta è giusta o meno; e, se sbagliata, spiega come correggerla.

```
  10 LET v=10
  20 LET c=0
  30 PRINT "Ehila'! Sono Sinclai
r Spectrum. Tu come ti chiami?"
  40 INPUT n$
  50 PRINT "OK, ";n$;" sai risol
vere"
  60 PRINT "questo ?"
  70 LET x=INT (v*v*RND)
  80 LET y=INT (v*v*RND)
  90 IF x+y<=12 THEN GO TO 7*v
 100 LET x1=INT (x/v)
 110 LET x2=x-v*x1
 120 LET y1=INT (y/v)
 130 LET y2=y-v*y1
 140 IF x2+y2>=v THEN LET r=1
 150 PRINT x;"+";y;" = ? "
 160 INPUT a
 170 PRINT a;" ?"
 180 LET b=x+y-a
 190 PRINT ("esatto" AND b=0)+("
mi spiace, sbagliato" AND b<>0)
 200 IF b=0 THEN GO TO 190
 210 PRINT x2;"+";y2;" fa ";x2+y
2-r*v
 220 IF r=1 THEN PRINT "riporto
1"
 230 PRINT " poi ";x1;"+";y1;"+
il riporto" AND r=1;" fa ";x1+y1
+r
 240 PRINT "che fa ";x+y
```

Note al programma

1. La linea 30 serve a stabilire una certa confidenza. Domanda all'operatore di scrivere il proprio nome. La linea 50 usa il nome per porre una domanda.
2. La linea 160 è dove l'operatore dice al computer quale pensa che sia la risposta.
3. Dalla linea 200 in avanti, in caso di risposta sbagliata si spiega all'operatore come eseguire la somma.
4. La difficoltà principale in questo genere di programmi "educativi" è che tendenzialmente richiedono lunghe parti di testo scritto. Si può molto migliorare questo programma, scegliendo vari livelli di difficoltà, dando maggiori avvertimenti se c'è qualche cosa che non va nei calcoli dell'operatore, proteggendo meglio il programma contro l'azionamento del tasto sbagliato, ecc. Inoltre, naturalmente, si potrebbero inserire delle operazioni più interessanti di banali addizioni. Tuttavia, la maggior parte dei principi basilari sono ben tratteggiati in questo programmino.
   Si possono inoltre comperare programmi "educativi" già preparati, pubblicizzati sulle riviste. La qualità di questi varia enormemente, e alcuni sono inferiori al mio!
5. Un buon programma non dovrebbe richiedere ogni volta la

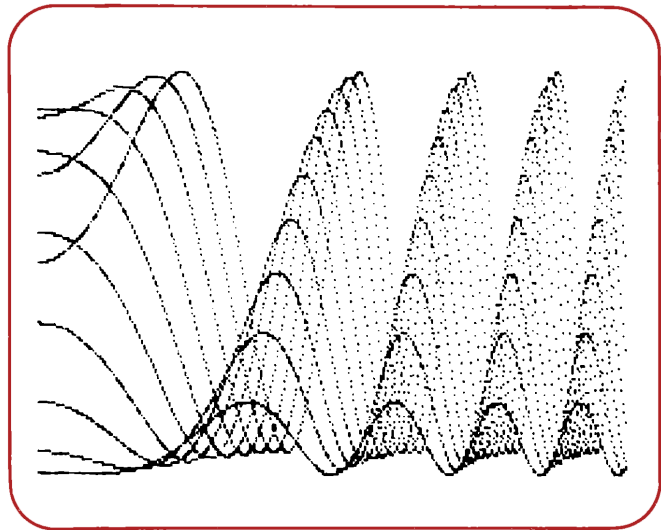
pressione di RUN per ripartire, ma dovrebbe domandare all'operatore se desidera un'altra esecuzione e comportarsi di conseguenza. È facile aggiungere qualche linea allo scopo.

Dimostrazione grafica I

Un assaggio della capacità grafica ad alta risoluzione dello Spectrum. Per essere un programma così corto, produce un'immagine notevolmente affascinante.

```
  1 BORDER 0: PAPER 0: CLS
 10 FOR j=0 TO 9
 20 INK 7-INT (j/2)
 30 FOR i=0 TO 510
 40 PLOT .5*i,(80+70*SIN (i*i/1
0000+j/2))*(1-j*j/100)
 50 NEXT i
 60 NEXT j
```

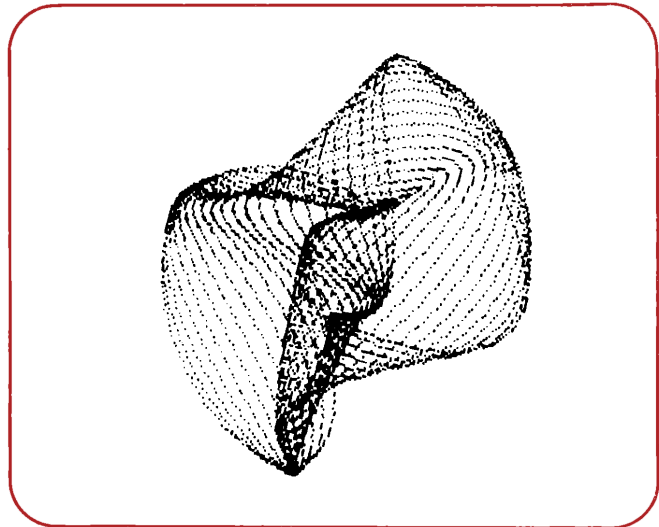
Note al programma

1. Benché i cambiamenti nel colore dei punti (INK) influenzino un intero blocco di 8 × 8 pixel, in questo caso l'effetto è abbastanza gradevole, e in pratica non si notano i blocchi.
2. Questo programma illustra un buon modo di ottenere interessanti figure ad alta risoluzione: sovrapporre diverse curve simili. Per "simili" intendo dire che ad ogni passo si apportano alla formula solo alcune modifiche che variano regolarmente: notate in questo caso come i valori di j modifichino la curva.

Dimostrazione grafica II

Questa volta il poverino continuerà a disegnare all'infinito, a meno che non gli togliate l'alimentazione o gli diciate di fermarsi.

```
   1 BORDER 0: PAPER 0: CLS : IN
K 7
   5 LET i=0
  10 PLOT 128+(30+50*COS (i/41))
*COS (i/40),88+(50+30*SIN (i/39)
)*SIN (i/40)
  15 LET i=i+1
  20 GO TO 10
```

Note al programma

1. Se questo programma vi piace, provate a cambiare i coefficienti della formula. In particolare il 41 e il 39 si possono trasformare in qualsiasi altro valore ragionevole, senza che la figura esca dallo schermo.
2. Se non vi soddisfa il bianco e nero, potete premere BREAK; poi INK con qualche valore; poi CONTINUE. I pixel seguenti verranno disegnati nel nuovo colore.

Fattori primi

Lo Spectrum è un calcolatore numerico piuttosto efficiente. Per convincervene, prendete un numero di 8 cifre e guardate con che rapidità questo programma vi darà i suoi fattori primi.

```
  10 INPUT k
  20 PRINT k;" = ";
  40 LET k0=k
  50 IF INT (k/2)*2=k THEN GO TO
 140
  60 LET n=3
  70 IF INT (k/n)*n=k THEN GO TO
 110
  80 IF n*n>k THEN GO TO 170
  90 LET n=n+2
 100 GO TO 70
 110 PRINT n;".";
 120 LET k=k/n
 125 IF k=1 THEN STOP
 130 GO TO 70
 140 PRINT "2.";
 150 LET k=k/2
 160 GO TO 50
 170 IF k=k0 THEN PRINT FLASH 1;
"numero primo"
 180 IF k<k0 AND k>1 THEN PRINT
k
```

Note al programma

1. INT (k/n) * n = k se e solo se k è divisibile per n. Questo criterio stabilisce i divisori.
2. Il programma prova a dividere k per 2 e per tutti i numeri dispari minori della radice quadrata di k. Se nessuno di questi è divisore, allora k è primo.
3. Se si trova un divisore, il programma divide k per questo valore, cerca poi un nuovo divisore della stessa grandezza. Se non lo trova, ne cerca uno maggiore.
4. La macchina impiega circa 35 secondi per un numero primo di 8 cifre, altrimenti meno. (Il tempo citato vale per 11111117: ovviamente cambia per altri numeri).
5. Modifiche: costruite un ciclo per ottenere i fattori primi di k, k + 1, k + 2,... e così via all'infinito.
6. k deve essere al massimo di 8 cifre, perché lo Spectrum non tratterà numeri più elevati con sufficiente precisione.
7. Questo programma si può rendere più efficiente dal punto di vista matematico, per esempio escludendo i multipli di 3 e 5 dai divisori che si stanno provando. Potete far sì che questo *acceleri* il programma? Il prezzo da pagare per l'efficienza matematica sono le maggiori dimensioni del programma: il guadagno riesce a superare la perdita?

Equazioni di terzo grado

Tutti hanno dei programmi per risolvere equazioni di secondo grado. Che ne dite di questo?

Trova tutte le radici reali di una cubica $ax^3 + bx^2 + cx + d = 0$, con una precisione accettabile.

```
  10 INPUT a,b,c,d
  50 LET b=b/a
  60 LET c=c/a
  70 LET d=d/a
  80 LET x=0
  90 LET g=2*x*x*x+b*x*x-d
 100 LET h=3*x*x+2*b*x+c
 110 IF h=0 THEN GO TO 200
 120 IF ABS (x-g/h)<1e-8 THEN GO
TO 300
 130 LET x=g/h
 140 GO TO 90
 200 LET x=x+1
 210 GO TO 90
 300 PRINT "x1= ";x
 400 LET a=b+x
 410 LET b=x*x+b*x+c
 420 LET d=a*a-4*b
 430 IF d<0 THEN PRINT "altre ra
dici immaginarie"
 440 IF ABS d<1e-7 THEN PRINT "p
ossibile instabilita' numerica"
 445 IF d<0 THEN STOP
 450 PRINT "x2= ";(-a+SQR d)/2
 460 PRINT "x3= ";(-a-SQR d)/2
```

Note al programma

1. Il programma calcola una radice dell'equazione tramite un processo iterativo noto come *metodo di Newton-Raphson*, che comporta l'assunzione di un valore di prova per x e successivi raffinamenti finché non si arriva sufficientemente vicino a una radice. Le linee 90-150 eseguono questo lavoro.
2. Se desiderate vedere come funziona questa iterazione, aggiungete

   ```
   131 PRINT x
   ```

   e guardate in che modo i numeri convergono alla radice.
3. Trovata una radice, il programma divide l'equazione per questo valore, per ottenere un'equazione di secondo grado da risolvere con la formula alle linee 400-460.
4. Se questa equazione di secondo grado non ha radici reali, si applica la linea 430; il programma va poi in errore alla linea 450, ma ciò non produce alcun danno.
5. Una precisazione: per alcune equazioni gli errori della parte aritmetica si accumulano in modo eccessivo e il programma potrebbe affermare "Altre radici immaginarie" quando in effetti questo non è vero. La linea 440 avverte l'operatore di questa possibilità. Il problema è che il segno di d è critico, e se d è troppo vicino a 0, gli errori possono avere un effetto disastroso. La maggior parte dei metodi numerici ha problemi analoghi, e una gran parte della disciplina nota come analisi numerica se ne occupa in dettaglio.

6. Per esempio, provate a inserire a = 4, b = —8, c = 5, d = —51. Dovreste ottenere x1 = 3, altre radici immaginarie. Provate ora con a = 4, b = —16, c= —5, d = 51: questa volta si ha x1 = 3, x2 = 2.6213203, x3 = —1.6213203.

Macchinetta mangiasoldi

Anche voi potete avere un gioco d'azzardo elettronico.

```
  10 LET c=0
  20 DIM a(3)
  30 FOR t=1 TO 3
  40 LET r=INT (3*RND)
  50 LET a(t)=r
  60 LET i=5
  70 LET j=10*t-5
  80 GO SUB 300
  90 NEXT t
 100 LET c=c-1
 110 IF a(1)=a(2) AND a(2)=a(3)
THEN GO TO 700
 120 GO SUB 1000
 130 INPUT "stop?";r$
 140 IF r$="s" THEN STOP
 150 GO TO 30
 300 PRINT INK 2*r;AT i,j;"▄▀▄▀";A
T i+1,j;"▄▀▄▀"
 310 BEEP .1,3*r
 320 RETURN
 700 LET c=c+9
 720 GO TO 120
1000 IF c>=0 THEN PRINT AT 18,5;
"Avete vinto ";c*1000;" lire "
1010 IF c<0 THEN PRINT AT 18,5;"
Avete perso ";-c*1000;" lire "
1020 RETURN
```

Note al programma

1. Vengono mostrati sul video tre quadratini colorati. Se sono tutti uguali, vincete 9000 lire, altrimenti ne perdete 1000. Per giocare premete qualsiasi tasto eccetto "s" — consigliamo ENTER, dal momento che vi risparmia una battuta. Premete "s" per finire il gioco.
2. In media dovreste andare alla pari. È un risultato migliore di quello conseguibile con le macchinette normali.
3. Notate i sottoprogrammi per disegnare i tre possibili tipi di quadrati.
4. Notate il buon uso dello schermo: si ottiene un display centrale di dimensioni ragionevoli.

Fattoriali oscuri

Lo Spectrum vi permetterà di chiamare un sottoprogramma dal suo stesso interno. Questa tecnica è nota come programmazione ricorsiva. Cercate di stabilire come funziona questo programma.

```
 10 LET f=1
 20 INPUT n
 30 LET m=n
 40 GO SUB 100
 50 PRINT m;" fattoriale e' ";f
 70 STOP
100 IF n<=1 THEN RETURN
110 LET f=f*n
120 LET n=n-1
130 GO SUB 100
140 RETURN
```

Calendario

In che giorno della settimana siete nati? Che giorno era quando morì Anna Bolena? Questo programma accetta in ingresso una data g. m. a., dove g è il numero del giorno, m il mese e a l'anno (p.e. 23.7.1066), e calcola che giorno sia (fosse, sarà).

```
  10 LET a$="033614625035"
  20 LET b$="domlunmarmergiovens
ab"
  30 INPUT g
  40 PRINT g;".";
  50 INPUT m
  60 PRINT m;".";
  70 INPUT a
  80 PRINT a;" e' ";
  90 LET z=a-1
 100 LET c=INT (z/4)-INT (z/100)
+INT (z/400)
 110 LET x=a+g+c+VAL a$(m)-1
 120 IF m>2 AND (a=4*INT (a/4) A
ND a<>100*INT (a/100) OR a=400*I
NT (a/400)) THEN LET x=x+1
 130 LET x=x-7*INT (x/7)
 140 PRINT b$(3*x+1 TO 3*x+3)
```

Note al programma

1. La linea 10 contiene una lista di dodici ''fattori correttivi mensili'' sotto la compatta forma di una stringa. La linea 100 calcola il mese nella lista come parte di una computazione spiegata alla nota 4.
2. La linea 20 usa un espediente simile per semplificare la routine di stampa. Notate anche come la linea 140 seleziona il gruppo di tre lettere richiesto.
3. La linea 100 calcola quanti anni bisestili sono passati dall'anno dato. Ricordate: i multipli di 4 sono anni bisestili, ma non i multipli di 100 se non sono anche multipli di 400.
4. La linea 110 è il punto essenziale della computazione. L'idea è di contare il numero di giorni trascorsi fino ad una data di riferimento (essenzialmente arbitraria), ma di risparmiare spazio eliminando i multipli di 7 dal momento che questi non influenzeranno il giorno della settimana. Così il numero di anni, a, va moltiplicato per 365; ma 365 = 52 * 7 + 1, così invece di 365 * a usiamo a. Le particolarità delle lunghezze dei mesi vengono prese in considerazione da a$ (m). Notate l'uso di VAL per convertire un carattere di una sola cifra in un vero e proprio numero. Per calibrare il programma ho scelto una data della quale conoscessi il giorno della settimana (30.9.1981, era un Mercoledì) e questo mi ha dato la correzione terminale —1. La linea 120 controlla la computazione in gennaio o febbraio di un anno bisestile, dove altrimenti sarebbe errata.

Progetti

5. Non si tiene conto di una sottigliezza del calendario. Nel 1582 il calendario è cambiato dal Giuliano al Gregoriano. Il giorno dopo il 4.10.1582 è stato il 15.10.1582. Modificate il programma in modo che funzioni anche con date anteriori al 1582.
6. Il programma accetta anche delle date impossibili come —37.12.—992. Modificatelo perché accetti solo valori sensati.
7. Non funzionerà per le date Avanti Cristo. Modificatelo in modo

che funzioni anche in questo senso. NB: non c'è stato l'anno 0 tra l'1 AC e l'1 DC, anche se logicamente avrebbe dovuto esserci!

Conversione binario/decimale

I numeri binari usano solo zero e uno; invece di 0, 1, 2, 3, 4, 5, 6,... si hanno 0, 1, 10, 11, 100, 101, 110,... dove in generale un numero come 1101001 viene elaborato da destra a sinistra come

$$1 \times 1 + 0 \times 2 + 0 \times 4 + 1 \times 8 + 0 \times 16 + 1 \times 32 + 1 \times 64 = 105$$

avendo ogni cifra un peso doppio del precedente. I computer, naturalmente, usano i numeri binari per le operazioni interne (benché i moderni calcolatori in pratica usino delle sofisticate varianti).

I due programmi che seguono convertono un numero in ingresso da binario a decimale e viceversa.

Binario → decimale

```
10 INPUT a$
20 PRINT a$;
30 LET l=LEN a$
40 LET s=VAL a$(1)
50 FOR j=2 TO l
60 LET s=2*s+VAL a$(j)
70 NEXT j
80 PRINT " vale ";s;" in decim
ale"
```

Decimale → binario

```
10 LET c$=""
20 INPUT a
30 PRINT a;
40 LET d=INT (a/2)
50 LET r=a-2*d
60 IF r=0 THEN LET b$="0"
70 IF r=1 THEN LET b$="1"
80 LET c$=b$+c$
90 IF d=0 THEN GO TO 120
100 LET a=d
110 GO TO 40
120 PRINT " vale ";c$;" in bina
rio"
```

Note al programma

1. L'INPUT per una conversione binario/decimale deve essere una sequenza di 0 e 1, come 11000101011. Per quella decimale/binario deve essere un numero intero come 3427006.
2. I due programmi sono stati scritti per illustrare diversi approcci al problema della conversione. Sarebbe possibile renderli molto simili nella struttura.
3. Notate che, nella conversione decimale/binario, le linee 60-70 *non* si possono sostituire con

```
60 LET b$ = "r"
```

Si può evitare di avere due linee usando CHR$ oppure

```
60 LET b$ = ("1" AND r) + ("0" AND 1 — r)
```

che funziona per motivi collegati a come lo Spectrum gestisce la logica — Vedi *Manuale*, pag. 137. Oppure, ancora:

```
60 LET b$ = "0"
70 IF r = 1 THEN LET b$ = "1"
```

Colpite il mattone

Dal cielo cadono mattoni colorati. Spariscono se riuscite a colpirli con il vostro bastone. Se li mancate, si accumulano. Il gioco termina quando le pile di mattoni diventano troppo alte. Quanti riuscite a catturarne?

```
   1 BORDER 1: PAPER 7: CLS
  10 LET h=11
  20 LET c=0
  30 DIM a(5)
  40 LET p=INT (5*RND+1)
  50 LET j=5*p
  60 FOR i=1 TO 21-a(p)
  70 PRINT AT i,j; INK 6*RND; IN
VERSE 1;" "
  80 BEEP .02,35-1.9*i-j/2
  90 PRINT AT i-1,j;" "
 100 LET m$=INKEY$
 110 IF m$="5" THEN LET h=h-1
 120 IF m$="8" THEN LET h=h+1
 130 IF h<1 THEN LET h=1
 140 IF h>26 THEN LET h=26
 150 PRINT AT 15,h;" "; INK 2; I
NVERSE 1;"   "; INVERSE 0;" "
 160 IF i=15 AND ABS (h+2-j)<=1
THEN LET c=c+1: GO TO 40
 170 NEXT i
 180 LET a(p)=a(p)+1
 190 IF a(p)=7 THEN PRINT AT 2,5
; INK 1;"Hai preso ";c;" mattoni
": STOP
 200 GO TO 40
```

Nota al programma Per muovere il bastone rosso a sinistra o a destra usate i tasti 5 e 8 marcati con la freccia.

Spirali e rosette

Ancora qualcosa sulla grafica, con un utile espediente per usare DRAW nel tracciamento di curve.

```
   1 BORDER 1: PAPER 2: CLS : IN
K 7
  10 FOR t=0 TO 4 STEP .5
  15 PLOT 128,88
  20 FOR y=0 TO 720
  30 LET x=y*PI/180
  40 LET r=1.5*x
  50 LET a=128+x*r*COS (x+t)
  60 LET b=88+4*r*SIN (x+t)
  70 DRAW a-PEEK 23677,b-PEEK 23
678
  80 NEXT y
```

Note al programma

1. "PI" alla linea 30 è il tasto M nel modo esteso, non le lettere P e I.
2. La linea 70 fa uso della variabile di sistema COORDS (vedi *Manuale* pag. 229) per calcolare l'esatto spostamento per tracciare una linea dalla posizione corrente alla successiva. In generale, se sostituite l'istruzione PLOT p, q con DRAW p — PEEK 23677, q — PEEK 23678, la macchina traccerà una linea retta fino alla successiva posizione PLOT. Esistono altri metodi adatti allo scopo, ma l'uso di COORDS evita il sommarsi degli errori, cosa che non avviene con altri metodi. Potete sperimentare quest'idea, anche senza comprendere appieno PEEK.
3. Per disegnare rosette invece di spirali, trasformate le linee 20 e 40 in:

```
20  FOR y = 0 TO 360
40  LET r = 20 + SIN (7 * x)
```

4. Fate qualche prova, sostituendo il "7" con altri valori, come 3, 4, 5, 6, 8, 9, 10.

Picasso

Questo programma non può aiutarvi a disegnare come Picasso, ma finché rimarrete sul colore 1, potrete avere un periodo blu anche voi! Il programma inizia chiedendo un ''modo''. Questo può essere ''d'' per *disegna*, ''c'' per *cancella*, oppure ''f'' per *fine*. In quest'ultimo caso il programma viene terminato.

Se inserite ''d'', il programma domanda un colore. Rispondete con il corrispondente numero (1 per il blu, 2 per il rosso, ecc.).

Sia per ''d'' che per ''c'', il programma ora richiede un'opzione, che deve essere un numero nell'intervallo 1-9. L'effetto di ogni singola opzione, è indicato di seguito. Notate che la parola ''disegna'' va sostituita con ''cancella'' se il modo è ''c''.

Opzione	*Azione*
1.	Disegna le scale orizzontale e verticale per riferimento. In ogni quinta linea e colonna appare un punto, due punti (una breve linea) ogni decima e una linea lievemente più lunga ad ogni centesima posizione.
2.	Rettangolo. Vi si richiederanno le colonne destra e sinistra e le linee inferiore e superiore in cui devono apparire i lati. Vi si domanderà poi se il rettangolo debba essere disegnato oppure usato come riferimento (vedi opzione 7).
3.	Circonferenza. Vi si domanderanno la colonna e la linea del centro e il raggio.
4.	Segmento circolare. Dovrete fornire le coordinate dei due estremi dell'arco di circonferenza e la massima distanza della curva dalla linea retta.
5.	Linea retta. Vi si domanderanno le coordinate dei due estremi della linea.
6.	Linea curva. Vi si domanderanno le coordinate delle due estremità della linea e la massima distanza della curva dalla retta (non tracciata) congiungente questi due punti.
7.	Ombreggiatura. Il programma vi domanderà se la forma interna al rettangolo di riferimento corrente sia da colorare o tratteggiare. Notate che il riferimento rettangolare corrente è l'ultimo disegnato o imposto come riferimento. In questo contesto la parola ''interno'' è usata in modo inclusivo. In altri termini, se disegnate un rettangolo e quindi richiamate l'opzione 7, questo rettangolo verrà ombreggiato, a meno che non ci sia un'altra figura al suo interno, nel qual caso capiteranno cose strane. Se scegliete ''Tratteggio'', vi si domanderà la distanza tra ogni linea, e se il tratteggio debba essere rettilineo o circolare. Se deve essere circolare, vi si domanderà la massima distanza tra la curva e l'immaginaria retta congiungente i due estremi. (Inserite in precedenza l'opzione 2).
8.	Memorizza. Si scarica su nastro la figura. Dovrete assegnare un nome.
9.	Caricamento. Si ricarica una figura dal nastro. Vi si richiederà di fornire il nome relativo.

Non è necessario implementare l'intero programma tutto in una volta. I sottoprogrammi che iniziano alle linee 1000, 2000, 3000, ecc., trattano le opzioni 1, 2, 3, ecc. rispettivamente. Se desiderate iniziare con qualcosa di semplice e disegnare solo rettangoli e circonferenze, non è necessario codificare le linee da 4000 in avanti. Naturalmente in questo caso non richiedete le opzioni 4-9, oppure otterrete un messaggio d'errore. C'è un'altra cosa a cui fare attenzione: le routine che iniziano a 4000, 6000 e 7000 chiamano tutte la routine a 9500.

```
  10 INPUT "Scegliere un modo(f=
fine;c=cancella;d=disegna):";m$
  20 IF m$="f" THEN STOP
  30 IF m$="c" THEN OVER 1
  40 IF m$="d" THEN OVER 0: INPU
T "colore:";c: INK c
  50 INPUT "opzione:";op
  60 GO SUB 1000*op
  70 GO TO 10
1000 FOR x=0 TO 255 STEP 5
1010 PLOT x,0
1020 IF x/10=INT (x/10) THEN PLO
T x,1
1030 IF x/100=INT (x/100) THEN P
LOT x,2
1040 NEXT x
1050 FOR y=0 TO 175 STEP 5
1060 PLOT 0,y
1070 IF y/100=INT (y/100) THEN P
LOT 2,y
1080 IF y/10=INT (y/10) THEN PLO
T 1,y
1090 NEXT y
1100 RETURN
2000 INPUT "Colonna sinistra del
 rettangolo:";cs
2010 INPUT "Colonna destra:";cd
2020 INPUT "Fila inferiore:";fi
2030 INPUT "Fila superiore:";fs
2040 INPUT "disegno (d) o riferi
mento (r):";m$
2050 IF m$="r" THEN RETURN
2060 PLOT cs,fi
2070 DRAW 0,fs-fi
2080 DRAW cd-cs,0
2090 DRAW 0,fi-fs
2100 DRAW cs-cd,0
2110 RETURN
3000 INPUT "centro della circonf
erenza (colonna, fila):";cc,fc
3010 INPUT "raggio:";r
3020 CIRCLE cc,fc,r
3030 RETURN
4000 INPUT "primo angolo del seg
mento circolare;(colonna,fila):"
;c1,f1
4010 INPUT "secondo angolo;(colo
nna,fila):";c2,f2
4020 INPUT "max distanza tra cur
va e linea:";d
4030 GO SUB 9500
4040 PLOT c1,f1
4050 DRAW c2-c1,f2-f1
4060 DRAW c1-c2,f1-f2,a
4070 RETURN
5000 INPUT "primo estremo della
linea;(colonna,fila):";c1,f1
5010 INPUT "secondo estremo;(col
onna,fila):";c2,f2
5020 PLOT c1,f1
```

```
5030 DRAW c2-c1,f2-f1
5040 RETURN
6000 INPUT "primo estremo della
linea curva;(colonna,fila):";c1,
f1
6010 INPUT "secondo estremo;(col
onna,fila):";c2,f2
6020 INPUT "max distanza dalla l
inea retta:";d
6030 GO SUB 9500
6040 PLOT c1,f1
6050 DRAW c2-c1,f2-f1,a
6060 RETURN
7000 INPUT "colorare (col) o tra
tteggiare (tratt):";m$
7010 IF m$="col" THEN LET s=1: L
ET a=0: GO TO 7070
7020 INPUT "larghezza del tratte
ggio:";s
7030 INPUT "diritto (d) o curvo
(c):";m$
7040 IF m$="d" THEN LET a=0: GO
TO 7070
7050 INPUT "max distanza dalla l
inea retta:";d
7070 FOR l=fi TO fs STEP s
7080 FOR c=cs TO cd
7090 IF POINT (c,l)=1 THEN GO TO
 7120
7100 NEXT c
7110 GO TO 7190
7120 LET c1=c
7130 FOR c=cd TO cs STEP -1
7140 IF POINT (c,l)=1 THEN GO TO
 7160
7150 NEXT c
7160 LET c2=c
7170 PLOT c1,l
7175 IF m$="c" THEN LET f2=l: LE
T f1=l: GO SUB 9500
7180 DRAW c2-c1,0,a
7190 NEXT l
7200 RETURN
8000 INPUT "attribuite un nome a
lla figura da memorizzare sul na
stro:";f$
8180 SAVE f$SCREEN$
8190 RETURN
9000 INPUT "nome della figura da
 caricare dal nastro:";f$
9010 LOAD f$SCREEN$
9020 RETURN
9500 LET ll=SQR ((c2-c1)*(c2-c1)
+(f2-f1)*(f2-f1))
9510 LET a=ASN (2*ll*d/(d*d+ll*l
l))*2
9520 RETURN
```

Modifiche e miglioramenti

1. Al momento non ci sono test per assicurarsi che l'utente non cerchi di disegnare fuori dallo schermo. Inseritene.
2. Alcuni comandi hanno l'effetto di distruggere la scala orizzontale, così siete costretti a ricrearla tramite l'opzione 1. Come potreste evitare questo problema?
3. Che dire di una routine per costruire un tratteggio verticale? (O persino diagonale?).

Linette

Non si gioca la comune roulette al Casinò di Argento Vivo Harold, si gioca la Linette. È una roulette giocata in linea retta.

```
  10 LET w=100000
  20 INPUT "Faites vos jeux ";p$
  30 CLS
  40 INPUT "Quanto puntate? ";p
  50 LET w=w-p
  60 FOR n=0 TO 9
  70 PRINT AT 10,2*n+6;CHR$ (48+
n)
  80 NEXT n
  90 LET r=INT (5*RND)+5
 100 LET d=INT (10*RND)
 110 LET r=r*10+d
 120 FOR n=0 TO r
 130 LET x=n-10*INT (n/10)
 140 PRINT AT 10,2*x+6;CHR$ 143
 145 BEEP .05,x
 150 PRINT AT 10,2*x+6;CHR$ (48+
x)
 160 NEXT n
 170 PRINT AT 10,2*x+6; FLASH 1;
CHR$ (48+x)
 180 IF p$(1)="p" THEN GO TO 210
 190 IF p$(1)="d" THEN GO TO 220
 200 IF VAL p$(1)=d THEN LET w=w
+10*p
 205 GO TO 240
 210 IF INT (d/2)=d/2 THEN LET w
=w+2*p
 215 GO TO 240
 220 IF INT (d/2)<>d/2 THEN LET
w=w+p*2
 240 PRINT AT 16,9;w;" gettoni "
 250 IF w>=0 THEN GO TO 20
 260 PRINT "Il gorilla di Harry
sara'in girodomani mattina"
```

Note al programma

1. Dopo RUN, scegliete "pari" o "dispari", o un numero tra 0 e 9, battendo sulla tastiera il vostro gioco. (Qualsiasi parola che inizia con p viene letta "pari" e qualsiasi parola che inizia con d viene letta "dispari")
2. A questo punto dite quanto state puntando, inserendo un numero. Si inizia avendo a disposizione 100 000 lire, predisposte alla linea 10.
3. Se vincete, ottenete il doppio della puntata, su pari o dispari; 10 volte la puntata su un numero. Teoricamente questo produce un gioco "onesto".
4. Potete puntare più di quanto possedete; ma attenzione nel caso perdiate!
5. Per fermare il gioco, su un'INPUT di caratteri, battete DELETE e poi STOP. Su un'INPUT numerico, basta STOP.

Linette circolare

Contemporaneamente, a due isolati di distanza, il rivale di Harold, Samuel Testone, ha avuto un'idea originale... Questa è una Linette giocata su un cerchio. Hmmm...

```
  10 LET w=100000
  30 CLS
  40 CIRCLE 123,91,84
  50 CIRCLE 123,91,60
  60 FOR n=0 TO 9
  70 PRINT AT 10+9*COS (n*PI/5),
15+9*SIN (n*PI/5);CHR$ (48+n)
  80 NEXT n
  82 INPUT "Faites vos jeux, gen
te!";p$
  84 INPUT "Quanto diavolo volet
e puntare?";p
  86 LET w=w-p
  90 LET r=INT (5*RND)+5
 100 LET d=INT (10*RND)
 110 LET r=r*10+d
 120 FOR n=0 TO r
 130 LET x=n-10*INT (n/10)
 140 PRINT AT 10+9*COS (x*PI/5),
15+9*SIN (x*PI/5);CHR$ 143
 145 BEEP .05,x
 150 PRINT AT 10+9*COS (x*PI/5),
15+9*SIN (x*PI/5);CHR$ (48+x)
 160 NEXT n
 170 PRINT AT 10+9*COS (x*PI/5),
15+9*SIN (x*PI/5); FLASH 1;CHR$
(48+x)
 180 IF p$(1)="p" THEN GO TO 210
 190 IF p$(1)="d" THEN GO TO 220
 200 IF VAL p$(1)=d THEN LET w=w
+10*p
 205 GO TO 240
 210 IF INT (d/2)=d/2 THEN LET w
=w+2*p
 215 GO TO 240
 220 IF INT (d/2)<>d/2 THEN LET
w=w+p*2
 240 PRINT AT 16,9;w;" gettoni "
 250 IF w>=0 THEN GO TO 82
 260 PRINT FLASH 1;"Girate al la
rgo da Sammy il      Pescecane"
```

Note al programma

1. Le istruzioni per il gioco sono identiche a quelle della Linette del programma precedente.
2. "PI" è il tasto M in modo esteso.
3. Samuel Testone è un po' più volgare di Argento Vivo Harold.

Morse automatico

Ecco un esempio del modo di usare BEEP. Il programma accetta un messaggio dalla tastiera e lo trasforma in codice Morse. Il messaggio e il codice appaiono sullo schermo; simultaneamente dall'altoparlante dello Spectrum vengono prodotti i suoni, di diversa durata, corrispondenti a punti e linee.

```
   5 CLS
  10 INPUT "introdurre il messag
gio";m$
  20 FOR i=1 TO LEN m$
  30 LET c=CODE m$(i)
  40 IF c<32 OR c>32 AND c<65 OR
 c>90 AND c<97 OR c>122 THEN GO
TO 120
  50 IF c>96 THEN LET c=c-32
  60 LET c=c-64
 100 IF c=-32 THEN PAUSE 40: PRI
NT "  "
 110 IF c>0 THEN PRINT CHR$ (c+6
4);"  ";: GO SUB 200
 120 NEXT i
 130 STOP
 200 LET k$=a$(c)
 210 LET t=1
 220 IF k$(t)="1" THEN PRINT INK
 6;".";: BEEP .1,10
 230 IF k$(t)="2" THEN PRINT INK
 3;"_";: BEEP .3,10
 240 IF k$(t)=" " OR t=4 THEN PR
INT " ": RETURN
 250 LET t=t+1: GO TO 220
 500 REM riemp. matrice
 510 DIM a$(26,4)
 520 FOR r=1 TO 26
 530 INPUT a$(r)
 540 NEXT r
```

Note al programma

1. Per costruire la matrice a$ che contiene il codice Morse, è necessario iniziare con GO TO 500. Inserite quindi 26 stringhe di 1 e 2: queste costituiscono il codice Morse per le lettere A-Z, ove 1 sta per punto, 2 per linea. Ecco di seguito le stringhe:

Lettera	Stringa	Lettera	Stringa
A	12	N	21
B	2111	O	222
C	2121	P	1221
D	211	Q	2212
E	1	R	121
F	1121	S	111
G	221	T	2
H	1111	U	112
I	11	V	1112
J	1222	W	122
K	212	X	2112
L	1211	Y	2122
M	22	Z	2211

Non inserite le lettere: sono presenti solo per ricordarvi dove siete arrivati.

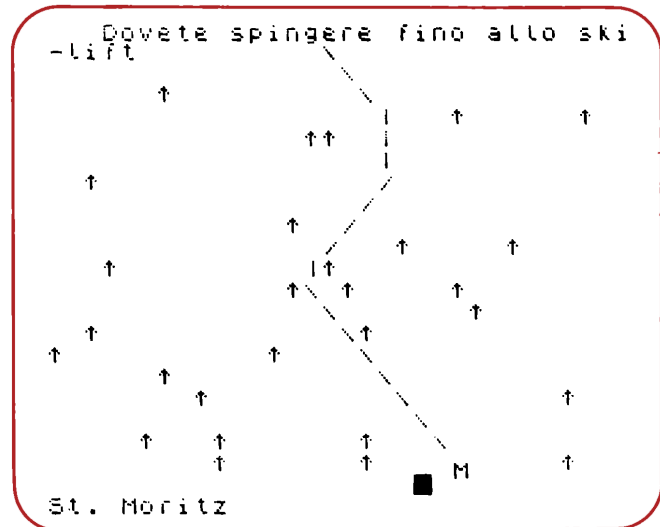
2. Dopo aver costruito a$, premete GO TO 5. *Non* premete RUN — cancellereste le variabili appena introdotte. Non usate mai RUN su questo programma.
3. Battete un messaggio, quando vi verrà richiesto. Se è più lungo di 22 caratteri, dovrete usare SCROLL. Il programma tratta nello stesso modo le lettere maiuscole e minuscole, e ignora qualsiasi altra cosa. Per esempio potreste battere ''Ciao''.
4. Ora il computer mostrerà il messaggio in lettere e in Morse; e riprodurrà con BEEP i punti e le linee.
5. Notate come viene usata la matrice di stringhe per contenere il codice Morse. Questo è un utilizzo molto comune per le matrici: come tabella di consultazione per la conversione da un sistema di codice a un altro.

Progetto

Modificate il programma in modo che possa accettare anche i numeri oltre che le lettere. Il codice Morse per i numeri da 0 a 9, nell'ordine, è:

—————, ·————, ··———, ···——, ····—, ·····, —····, ——···, ———··, ————·

St. Moritz

Dovete sciare attraverso la foresta, evitando gli alberi, per arrivare allo ski-lift. Voi siete una "M" (vista aerea di uno sciatore rannicchiato, capito?) e lo ski-lift è una macchia nera (non so perché!). È sempre discesa così, se non fate niente, cadete semplicemente dall'alto fino in fondo, o finché non incontrate un albero. Premete una "z" per curvare a sinistra, una "m" per andare a destra.

```
   1 CLS : RANDOMIZE
   2 LET n=30: LET sc=14
  10 FOR i=1 TO n
  20 LET r=INT (RND*18)+3
  30 LET c=INT (RND*32)
  40 PRINT AT r,c;"↑"
  50 NEXT i
  60 PRINT AT 21,20;"█"
  70 PRINT AT 0,sc;"M"
  80 PAUSE 200
  90 FOR r=1 TO 20
 100 LET d$=INKEY$
 103 IF d$="" THEN PRINT AT r-1,
sc;"|"
 104 IF d$="m" THEN PRINT AT r-1
,sc;"\": LET sc=sc+1
 105 IF d$="z" THEN PRINT AT r-1
,sc;"/": LET sc=sc-1
 120 IF SCREEN$ (r,sc)="↑" THEN
PRINT AT r,sc;"*crash": GO TO 20
0
 130 PRINT AT r,sc;"M"
 135 PAUSE 2
 140 NEXT r
 150 IF sc<22 AND sc>18 THEN PRI
NT FLASH 1;AT 0,2;"grande !!!":
GO TO 200
 160 PRINT AT 0,3;"Dovete spinge
re fino allo ski-lift"
 200 INPUT "un altro giro?";r$
 210 IF r$="si" THEN GO TO 1
 220 CLS : PRINT AT 10,2;"Et mai
ntenant l'apres-ski..."
```

Modifiche e miglioramenti

1. Gli alberi potrebbero avere un aspetto più simile ad alberi. Provate a disegnare un abete o a inventare una conifera: a questo scopo date un'occhiata anche al programma che segue. Visto che ci siete, che ne dite di dare una doppia scia allo sciatore; e *fate qualcosa* per quell'orribile ski-lift.
2. Il numero di alberi è stato fissato a 30. Fate sì che l'utente possa variarlo per avere un gioco più o meno difficoltoso.
3. Alla linea 135 del ciclo c'è un PAUSE 2 per dare una velocità di discesa ragionevole. Potreste permettere all'utente di modificarlo. (Non intendo direttamente: costruite una serie di possibili valori di PAUSE e permettete all'utente di introdurre, per esempio, un tasso di difficoltà da 1 a 5 che selezioni un diverso valore di PAUSE).
4. Perché non modificare automaticamente la difficoltà, inserendo l'intero programma in un ciclo e aumentando il numero di alberi e/o la velocità di discesa ogni volta che l'utente completa una discesa con successo?

Costruzione di caratteri

Questo programma vi permette di disegnare un carattere grafico in dimensioni ingrandite sullo schermo TV, e lo inserisce nel computer come il carattere-utente corrispondente a una lettera di vostra scelta. Stampa anche il contenuto degli otto byte che definiscono il carattere, per un successivo riutilizzo in altri programmi.

```
  10 DIM k(8,8)
  20 FOR i=8 TO 15
  30 PRINT AT i,12;"........"
  40 NEXT i
  50 LET x=0: LET y=0
  55 GO SUB 500
  60 INPUT "valore del pixel ";v
  65 IF v<>0 AND v<>1 THEN GO TO
 60
  70 LET k(x+1,y+1)=v
  80 PRINT AT x+8,y+12; INVERSE
v;" "
  90 LET y=y+1
 100 IF y=8 THEN LET y=0: LET x=
x+1
 110 IF x=8 THEN LET x=0: LET y=
0: GO TO 200
 120 GO TO 55
 200 INPUT "carattere soddisface
nte?";q$
 210 IF q$="" THEN GO TO 200
 220 IF q$(1)="s" THEN GO TO 600
 230 INPUT "usare i tasti con la
 freccia per muovere il cursore"
;j$
 235 GO SUB 500
 240 IF INKEY$<>"" THEN GO TO 24
0
 250 IF INKEY$="" THEN GO TO 250
 260 LET i$=INKEY$
 270 IF i$="0" OR i$="1" THEN LE
T k(x+1,y+1)=VAL i$: PRINT AT x+
8,y+12; INVERSE VAL i$;" ": GO T
O 200
 275 PRINT AT x+8,y+12; INVERSE
k(x+1,y+1);" "
 280 IF i$="5" THEN LET y=y-1+(y
=0)
 290 IF i$="6" THEN LET x=x+1-(x
=7)
 300 IF i$="7" THEN LET x=x-1+(x
=0)
 310 IF i$="8" THEN LET y=y+1-(y
=7)
 320 GO SUB 500
 330 GO TO 240
 500 PRINT AT x+8,y+12; FLASH 1;
 INK 2;"*": RETURN
 600 INPUT "quale lettera?";f$
 610 FOR n=1 TO 8
 620 LET t=k(n,1)
 630 FOR j=2 TO 8
 640 LET t=2*t+k(n,j)
 650 NEXT j
 660 POKE USR f$+n-1,t
 670 NEXT n
 700 PRINT AT 17,0;"questo e' l'
aspetto definitivo: ";AT 18,15;C
HR$ (CODE f$+47)
 710 PRINT "byte: "
 720 FOR n=1 TO 8
 730 PRINT PEEK (USR f$+n-1);" "
;
 740 NEXT n
```

Note al programma

1. Inizialmente, lo schermo mostra una griglia di 8 × 8 punti, e un cursore lampeggiante; vi si chiede il valore del pixel. Inserite 0 o 1: inserendo 0 si cancella il punto; se introducete 1, si annerisce il relativo quadrato. Il cursore percorre automaticamente l'intera matrice, riga per riga; i vostri dati costruiscono la prima prova del carattere.
2. Vi si chiede ora "Carattere soddisfacente?". Se premete "s" — o qualsiasi cosa che inizi con s — il carattere viene accettato dalla macchina e si passa alla fase successiva; se rispondete qualsiasi altra cosa, appare il messaggio "Usare i tasti con la freccia per muovere il cursore". Dovete ora premere ENTER per fare riapparire il cursore, che si può controllare dalla tastiera usando i tasti 5, 6, 7, 8 per muoverlo nella direzione delle frecce. Quando avrete raggiunto la posizione desiderata, un ingresso di 0 o 1 da tastiera cancellerà o annerirà il quadrato corrispondente. Riappare il messaggio "Carattere soddisfacente" e, se volete, potete apportare ulteriori modifiche.
3. Una volta costruito il carattere *giusto*, e premuto il tasto "s", la macchina vi domanderà a quale lettera desiderate far corrispondere il vostro carattere. Ricordate che a ogni carattere utente si può accedere da tastiera usando una lettera (eccetto v, w, x, y, z): dovete decidere quale. Per esempio, potreste scegliere "s": a questo punto il vostro nuovo carattere è memorizzato in un luogo tale da essere riprodotto dalla "s" nel modo grafico. Potete anche richiamarlo per mezzo di un'istruzione CHR$, in questo caso CHR$ (162).
4. Il computer elenca anche gli otto byte corrispondenti alle linee del carattere: questo semplifica la sua successiva ricostruzione, annotando i numeri e riponendoli al loro posto tramite delle POKE, come descritto nel Capitolo 23.
5. Da ora in poi, finché non spegnerete, il vostro nuovo carattere sarà memorizzato come "s" in modo grafico. Potete caricare

altri caratteri in posizioni diverse, comandando una nuova esecuzione (RUN).

6. In modo da poter controllare la correttezza del vostro lavoro, ottenete una stampa di prova del vostro nuovo carattere. Se non vi piace, premete GO TO 200. Il carattere non viene stampato ma, come se aveste ripulito tutta l'area con il cursore, riappaiono i quadratini, e potete apportare nuove modifiche.

Tiro a segno

Al poligono di tiro in una giornata ventosa: che punteggio potete fare?

```
  10 LET d=100
  20 LET ven=40*(.5-RND)
  30 LET v=1000
  40 LET s=0
  50 LET c=0
 100 FOR i=1 TO 8
 110 CIRCLE 127,95,8*i
 120 NEXT i
 200 IF c=11 THEN STOP
 205 PRINT AT 20,0;"alzo=";
 210 INPUT a: PRINT a
 220 PRINT "deviazione=";
 230 INPUT dev: PRINT dev
 240 LET a=a*PI/180
 250 LET dev=dev*PI/180
 300 LET t=d/(v*COS a)
 310 LET h=v*t*SIN a-4.9*t*t
 320 LET k=ven*d/100+d*SIN dev
 330 LET h0=8*h+95+40*(.5-RND)
 340 LET k0=8*k+127+40*(.5-RND)
 350 IF h0<0 OR h0>170 OR k0<0 O
R k0>255 THEN GO TO 500
 360 GO SUB 390
 370 PRINT AT 20,0;"
    ",,"                ": REM 1
6+16 spazi
 375 LET c=c+1
 380 GO TO 200
 390 INK 3
 400 PLOT k0-8,h0: DRAW 16,0: PL
OT k0,h0-8: DRAW 0,16
 405 BEEP .1,5
 410 CIRCLE k0,h0,6
 415 IF c<1 THEN GO TO 480
 420 LET q=SQR ((k0-127)*(k0-127
)+(h0-95)*(h0-95))
 430 LET q=INT (q/8)
 440 LET q=100-10*q
 450 IF q<30 THEN LET q=0
 460 LET s=s+q
 470 PRINT AT 0,25;c;TAB 28;s
 480 INK 0
 490 RETURN
 500 PRINT AT 0,0;"fuori campo"
 510 PAUSE 50
 520 PRINT AT 0,0;"          ":
REM 10 spazi
 530 PRINT AT 20,0;"
    ",,"                ": REM 1
6+16 spazi
 535 PRINT AT 0,25;c
 540 LET c=c+1
 550 GO TO 200
```

Note al programma

1. Dopo RUN, vi si domanda d'inserire un alzo (in gradi). Un valore prossimo a zero è la scelta migliore, per esempio tra —2 e +2.
2. Vi si chiedono poi delle *deviazioni* laterali (per compensare un vento laterale). Queste dovrebbero essere entro circa —10 e +10. Non vi si dice in quale direzione sta soffiando il vento!
3. Avete un colpo di prova per rettificare il tiro, che non viene contato; poi 10 colpi. Tra i vari colpi si hanno piccole variazioni del vento.
4. Il punteggio, e il numero di colpi usati, vengono stampati in alto a destra — prima il numero di colpi, poi il punteggio.

Ian Stewart
Mathematics Institute, University of Warwick

Robin Jones
Computer Unit, South Kent College of Technology

Questo è il libro ideale per tutti coloro che hanno acquistato o contano di acquistare uno home computer ZX Spectrum. Prodotto in Gran Bretagna dalla Sinclair, lo Spectrum ha rapidamente raggiunto anche in Italia una grande popolarità, sia per la sua economicità, sia per l'ampio ventaglio di prestazioni che ne fanno un vero e proprio 'personal'.

Il libro descrive in termini chiari e comprensivi quelle caratteristiche dello ZX Spectrum che un utente principiante dovrebbe conoscere. Vi sono inclusi inoltre circa 30 listati di programmi 'preconfezionati' da copiare e da far 'girare' a discrezione dell'operatore, più qualcosa come altri 30 programmi nel corso del testo.

Ciò che si ottiene è una facile guida alla programmazione in BASIC, all'uso del colore, del suono e dell'animazione, alla grafica ad alta risoluzione, alla correzione dei programmi (debugging), all'elaborazione numerica e alle operazioni sulle stringhe.

Ogni argomento è trattato in sezioni autonome, in modo da permettere all'utente di trascorrere alcune ore con lo ZX Spectrum ottenendo risultati concreti. In più sono contenuti una varietà di programmi divertenti e ricreativi, che fanno ottimo uso della notevole gamma di possibilità del computer.

Il libro è destinato al vasto pubblico degli utenti attuali o potenziali dello ZX Spectrum. Per la semplicità dei temi esposti e per l'immediatezza d'uso, esso può essere utilizzato molto opportunamente anche nelle scuole sia dagli allievi — che in tal modo ricevono adeguate informazioni per eseguire le esercitazioni – sia dai docenti come ideale supporto per completare lezioni in forma dinamica o per sostituire o per integrare strumenti tradizionali.

0025367-4

Lire **18.000**
(IVA inclusa)

IAN STEWART
ROBIN JONES

IL PIACERE DI PROGRAMMARE CON LO ZX SPECTRUM